Dynamic FDG-PET of soft tissue sarcomas

Espen Rusten

Faculty of Mathematics and Natural Sciences
Department of Physics
Biophysics and Medical Physics

UNIVERSITY OF OSLO

September 2011

”If at first you don’t succeed, try, try again.

Then quit.

There’s no point being a damn fool about it”

W. C. Fields

År 2011

Dynamic FDG-PET of soft tissue sarcomas.

Espen Rusten

http://www.duo.uio.no/

Trykk: Reprosentralen, Universitetet i Oslo

Abstract

Positron emission tomography (PET) is a functional imaging modality used to visualize metabolically active tissues such as cancer. It is used clinically to determine the malignancy of lesions, scan for metastases as well as evaluating treatment response. The goal of this project was to create a computer program for analysis of dynamic FDG-PET images of patients with soft tissue sarcomas through the use of pharmacokinetics, and to assess the added value of dynamic PET compared to conventional, static PET.

FDG PET/CT scans from eleven patients with grade III or IV soft tissue sarcomas where obtained. Nine of the patients where liposarcomas, one was a schwannoma and one was a myxoid liposarcoma. Four of the patients were subjected to radiation therapy and their progress was recorded through three additional scans during and after the therapy. Time activity curves (TACs) revealed that the investigated sarcomas were a heterogeneous group with a wide spread of PET parameters. Mann Whitney statistics were used to estimate the predictive quality of the different parameters. It was found that none of the PET parameters have any predictive qualities when it comes to tumor type or location.

Compared to the direct PET image 45 minutes after injection, the pharmacokinetic parameter MR_{FDG} (metabolic rate of FDG) was found to have superior tumor-to-tissue contrast in one patient. The tumor-to-tissue ratio was found to amplify the vascular phase of the TAC's, although no additional diagnostic information could be detected. A direct early PET measurement was, together with the pharmacokinetic parameters K1 and the vascular fraction, found to be good descriptors of the vasculature. Patlak analysis was attempted but represented less contrast and more noise compared to the pharmacokinetic parameters. For the patients undergoing therapy some changes in the vasculature could be detected but all patients were concluded to be non-responders.

The current material indicates that dynamic PET may in some instances provide additional information compared to a static PET. However no capability to differentiate liposarcomas from other sarcomas could be found and it does not appear that PET is useful for monitoring early treatment response patients with soft tissue sarcomas. As no link to clinical data could be established the conclusion is that the extra time in the scanner could not be justified.

However, an extended study including subgroups of liposarcomas with different prognosis is recommended. In such a study the number of patients should be increased if results of statistical significance are desired. Improvements to the PET protocol are most likely an increased sampling rate for precise bolus detection while improvements to the software are likely to be found in improved plasma fitting and timing.

Preface

This thesis is submitted for the degree of Master of Science at the Department of Biophysics and Medical Physics and most of the practical work was performed at the Department of Medical Physics at the Norwegian Radium Hospital (Oslo University Hospital).

At the Department of Medical Physics I would like to thank the Head of Department Vidar Jetne for the use of his facilities, Ingerid Skjei Knudsen for sharing workspace with me, Jan Rødal for his help on Masterplan and of course Eirik Malinen, my supervisor, for encouragements, guidance and for tolerating my idiosyncrasies.

I would like to thank everyone at the Department of Biophysics and Medical Physics for activities and fun moments, and especially Lars Tore Gyland Mikalsen for discussions on pattern recognition, Erik Olai Pettersen for discussions on hypoxia, Magne Mørk Kleppestø for comments on commas and Børge Sæter for figuring out the figures.

Oslo, September 2011

Espen Rusten

Contents

1 Introduction

Cancer is a systemic disease characterized by uncontrolled cell proliferation combined with infiltrative growth and spread to other parts of the body (metastasis). It stems from changes in the normal cells (mutations) where the normal control mechanisms for growth and proliferation are suspended. Most functions in the cell, including the control mechanisms, are performed or stimulated by proteins which in turn are produced according to the genetic material (DNA) expressed. As such, from a genetic point of view, the cause of cancer is unrepaired DNA damage to certain gene sequences. Although the exact mechanisms differ there are generally three different types of genes that are mutated. Normally there are mutations in genes that mediate signals from cell surface to the cell nucleus, genes that control progression through the cell cycle and genes responsible for maintaining DNA integrity. The end result is cells independent of growth factor control and with a high mutation rate leading to evolutionary selection of aggressive cells.

In medical imaging both the internal structures and functions of the body can be visualized. Structure, like bones, muscles or soft tissues are usually imaged through the use of CT or MRI. MRI is an imaging modality using proton spin changes in an external magnetic field to describe water distributions in the body. CT is an imaging modality where x-ray attenuation is used to describe the electron density in the body. In functional imaging the metabolism or blood flow of a region is imaged and while conventional MRI and CT scanners can in some instances be used their mode of operation is slightly modified.

The most common functional imaging modalities are positron emission tomography (PET), using fluorodeoxyglucose (FDG) as tracer, for imaging metabolism and dynamic contrast-enhanced MRI (DCE-MRI) or dynamic contrast enhanced CT (DCE-CT) for imaging perfusion flow. The perfusion flow may be estimated from modeling the contrast agent's arrival and distribution in the region in question. In DCE-MRI the contrast agent is a paramagnetic molecule, like gadolinium, that produces a local magnetic field leading to locally enhanced spin relaxation rates. In DCE-CT the contrast agent consist of atoms with a high atomic number, like iodine, leading to a high probability of photoelectric interaction.

Sarcomas are an uncommon type of cancer (~1% of all cases) that originates from the connective tissue. Precise studies of particular subclasses of sarcomas have been hindered

since the incidence is low and there are about 50 different heterogeneous subclasses. This leaves room for an improvement in customized treatment through the use of an approach like functional imaging that does not measure a particular protein but rather measure a biological process.

Soft tissue sarcoma (STS) is the major sarcoma subgroup and a rather aggressive class of tumors with a 30% 5 year recurrence free survival rate reported (Schwarzbach, Hinz et al. 2005) . It is a heterogeneous group of tumors characterized by infiltrative local growth and metastases. Comparisons of the different imaging modalities and their abilities to define soft tissue sarcomas can be found in the literature (Karam, Devic et al. 2009), and it is stated that MRI should be preferred due to its superior detection of the infiltrative growth. Tumor size give a general idea of the malignancy but to get the full picture functional imaging should be used. The constant growth of the tumor requires an elevated metabolism and abnormal vasculature, both being markers for malignant tumors. Since tumors use a glycolytic metabolism PET has found a role for both for distinguishing benign lesions from malign lesions and for screening the whole body for metastases. In addition, soft tissue sarcomas are tumors relatively resistant to radiation therapy and the usual treatment is surgery, possibly combined with radiation therapy or chemotherapy (Jebsen, Trovik et al. 2008). Chemo therapy usually has adverse side effects. There is an ongoing evaluation of the use of PET scans to predict and evaluate therapeutic response for personalized treatment (Kasper, Dietrich et al. 2008).

PET scans may be conducted before, after and during treatment. Also it is desirable to investigate the possibility of extracting additional information from PET scans. The normal use of PET is to inject the radioactive tracer, wait typically 60 minutes and then detect the resulting distribution of the tracer. However, as the tracer is initially concentrated to a bolus in the blood stream it has been proposed that dynamic PET can be used in a similar way as dynamic contrast enhanced CT or MRI to describe the vasculature and the perfusion flow (Malinen, Rodal et al. 2011). The PET resolution is lower than both DCE-MRI and DCE-CT, but the tracers used for these modalities are associated with contrast agent induced nephropathy. PET requires injection of a radioactive tracer, but if a PET scan is to be undertaken anyhow and the PET resolution is acceptable, dynamic PET may be preferable.

Pharmacokinetics is the study of how administered substances are transported and stored in the body through absorption, distribution, metabolism and excretion. The use of a

compartment model to calculate pharmacokinetic parameters have been examined in the literature (Dimitrakopoulou-Strauss, Strauss et al. 2001). However, it is still uncertain which parameters are most important for tumor differentiation as well as the role of PET in pretreatment grading. In this thesis an evaluation of several PET parameters in a clinical setting is conducted. The major goals of this study are:

1. Creation of a computer program for reading and displaying dynamic PET images.
2. Analyzing the dynamic PET sequences of 11 patients.
3. Describing the FDG kinetics and the resulting pharmacokinetic parameters.
4. Classification of the tumors based on PET parameters.
5. Evaluation of the practical use of the results.

The computer program is implemented in Interactive Data Language (IDL) with a complete user interface and an observer structure. For all the patients regions of interest are defined, that is the tumor, the major arteries and some reference tissue. Finally the estimations of pharmacokinetic parameters are calculated, the result is inspected visually and classification is attempted through the use of distribution probabilities.

Part I
Theory and methods

2 Background

2.1 Cancer

This chapter is inspired by a standard text book on molecular Biology (Alberts, Johnson et al. 2008) and a paper on molecular mechanisms of sarcomagenesis (Matushansky and Maki 2005).

Glucose metabolism

Much of the arguments produced in this text center around the glycolytic metabolism in tumors and as such it is first important to understand the metabolism of normal cells. Glucose is the cells primary source of energy. It is distributed through the blood and the cellular uptake is regulated by insulin. In the cells the glucose is either stored as glycogen or metabolized to Adenosine triphosphate (ATP), which is a high energy molecule used in most biologic processes requiring energy. The processing of the glucose is divided into two parts. First the glucose is divided into two molecules of ATP and 2 molecules of pyruvate in a process called glycolysis. The pyruvate is then further processed in the Citric acid cycle, or Krebs cycle, releasing about 30 ATP. This final step is conducted inside the mitochondria which are the 'power plants' of the cells. This cycle does however require oxygen to function and this type of metabolism is thus termed aerobic. In the absence of oxygen the metabolism is termed anaerobic and glycolysis become the dominating energy producing process. The excess pyruvate is in this case fermented into lactate, and later lactic acid, which is transported out of the cells. From the cells it is transported by the blood stream to oxygenated cells or reverted to glucose through gluconeogenesis in the liver. The alternative source of energy in the body is fat, which is used for long time storage in specialized fat cells called adipocytes and released as fatty acids into the blood when required. In normal cells the mitochondria can also degrade fatty acids for energy and heart and skeletal muscles will prefer this over glucose. Neurons and red blood cells on the other hand only use glycolysis.

Carcinogenesis

In all complex life forms the (eukaryotic) cells live and proliferate under strict control governed by proteins produced in adequate amounts according to the cell's DNA. During the

cells life cycle the DNA is repeatedly damaged by chemical agents, virus, radiation and faulty cell replications. In most cases this damage is harmless as it affects an inactive part of the DNA sequence or is repaired before the damage is fixated. However, if the damage is in a DNA segment coding for an important protein and it is left uncorrected, the cell's natural properties are changed. The altered (mutated) cell may be unstable allowing mutations occur in it. If a certain combination of mutations occur in the same cell it might loose its ability to respond to the natural regulatory systems, and may begin an uncontrolled proliferation much like bacteria. In most cases the abnormal protein expression is detected by the cell itself and it enters apoptosis (controlled cellular suicide) but if the proteins that perform this control have been depleted the cell might survive. The resulting tissue created through this abnormal proliferation is called a tumor or neoplasm.

The development of a tumor is a complex chain of events and there is an ongoing effort to identify the proteins and genes involved. Figure 1 shows an overview of the proteins discussed in this chapter. Although the exact DNA composition is unique in each tumor disturbances in three signaling pathways usually occur. First, disturbances that lead to failure in halting the cell cycle before mitosis. This usually involves the p53-p21-cdk2 or the cdk4-Rb-E2F pathways, and facilitates proliferation and increases genomic instability. Second, disturbances that lead to failure in the mechanisms conserving DNA integrity, usually involving the p53 protein mentioned earlier. P53 is a protein that detects damage to the DNA, if such damage is detected it attempts to halt the cell cycle allowing time to repair the DNA (as described earlier). If the cell fails to halt the cell cycle or complete the repair, p53triggers apoptosis. If p53 is damaged, or suppressed, more uncorrected errors in replication will occur resulting in an increased mutation rate. Third, disturbances in the protein chain mediating signals from the cell surface receptors to the cell nucleus. For a cell to grow it need growth factors which in normal cases are supplied externally (in what is known as the ras-raf-map-myc pathway). Since the cells normally have to compete for growth factors, and in a tumor they are many and needy, there are not enough normal growth factors to sustain the tumor growth. For a cancer to develop it needs to alter some part of this chain to become independent of growth factors.

Usually a tumor that is confined to a distinct region is considered a benign tumor. However further mutations may give more aggressive cells an advantage as they live in a highly competitive unregulated environment. If the cells become invasive and start to produce new

colonies (metastases) the tumor is called malignant. This gradual development through multiple mutations is called carcinogenesis.

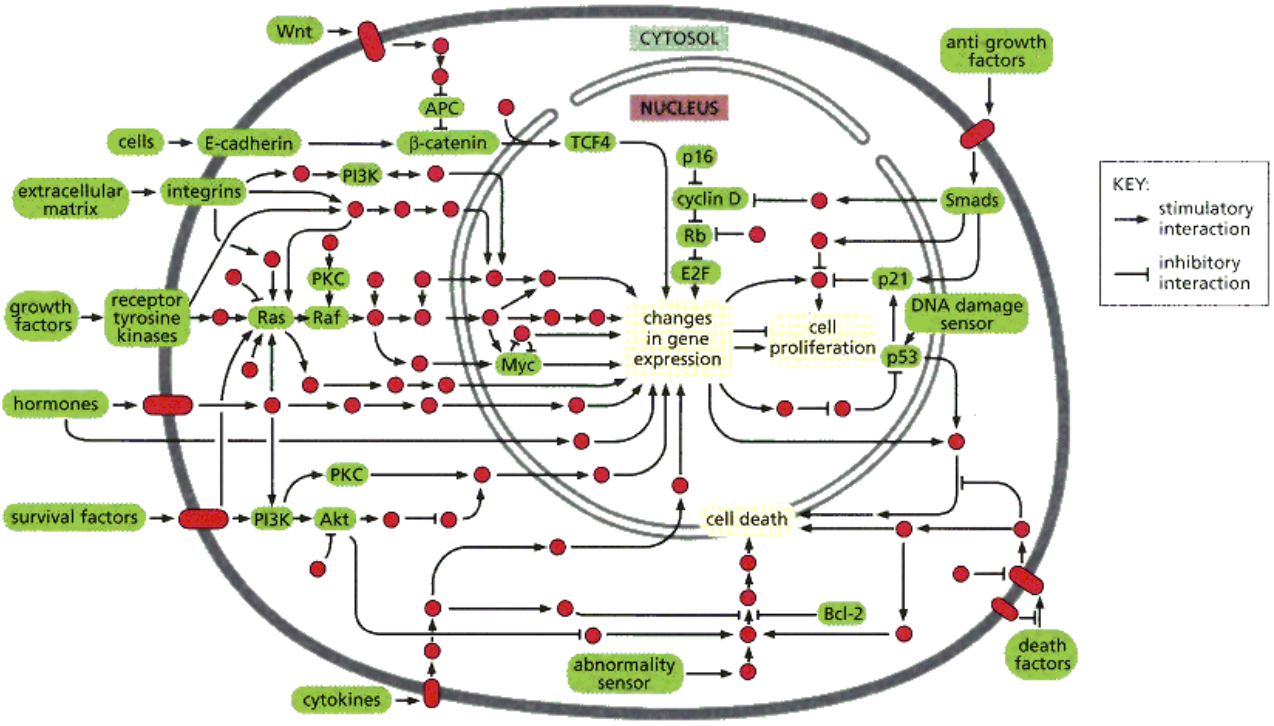

Figure 1. Schematic of the major signaling pathways relevant to human cancer cells. Green boxes are control mechanisms, red circles represent individual proteins. Lines indicate stimulatory or inhibitory interaction. (Alberts, Johnson et al. 2008)

Hypoxia and cancer metabolism

In solid tumors the increased cell density of the tissue leads to insufficient blood supply. The resulting lack of oxygen (hypoxia) will force the cells into anaerobic metabolism, which leads to a decrease in efficiency and production of lactic acid. This inhibits growth beyond a few million cells, and in this new stressful environment survival factors will be expressed. Two central survival factors are the transcription factor HIF 1α, hypoxia inducible factor, and the protein kinase Akt. It has been indicated that there is a correlation between hypoxia, HIF and expression of glut transporters and hexokinase (Zhao, Kuge et al. 2005). HIF transcribes the vascular endothelial growth factor that stimulates growth and division of blood vessels (angiogenesis), while Akt inhibit apoptosis and indirectly activates the protein mTor that upregulate the glucose (glut) transporters and glycolytic enzymes like hexokinase. The angiogenesis leads to increased supply of oxygen and glucose, however the new blood vessels are created in an erratic fashion and the blood supply is no longer constant. The upregulation

of the glycolysis on the other hand makes the cells resistant to hypoxia as the diffusion depth of glucose is deeper than that of oxygen. This is linked to the production of lactic acid in the hypoxic areas. As described earlier in this chapter, excess lactic acid is transported from the cells to surrounding normal cells with oxygen supply. Since the citric circle is more efficient than glycolysis they prefer to use lactate in their metabolism rather than glucose. This way the glucose is not absorbed in normal tissue and can diffuse into the tumor.

Upregulation of glycolysis is so potent that it is present in all tumors. This is called the 'Warburg effect', named after the German physiologist and biochemist Otto Heinrich Warburg (1883-1970). He discovered that cancer cells, even when oxygenated, depend primarily on glycolysis for their metabolism. Warburg concluded that cancer was caused by the deactivation of mitochondria and though this hypothesis later have been refuted the observation itself holds true. The Warburg effect is to a certain degree counter intuitive, as anaerobic glycolysis produces 2 ATP while the aerobic glycolysis (glycolysis + Krebs/Citric cycle) produce ~36 ATP. But it has been proposed that the anaerobic metabolism give a comparative advantage through its resistance to hypoxia, and the secretion of acids in itself is an advantage. The acidic environment is somewhat toxic to normal cells and help degenerate the intracellular matrix (Gatenby and Gillies 2004), which in turn indicate and aggressive and invasive tumor. The end result is that although carcinogenesis is a complex matter, glycolysis is usually an excellent marker for cancer detection.

Types of cancer

There are three general types of cancer classified after the type of tissue they originate from. First, carcinomas arise from epithelial tissue and are the most common type (approximately 90%) of human cancer. Second, leukemia and lymphoma arise from blood forming cells and cells of the immune system. Thirdly, sarcomas originate from muscle cells or connective tissue (mesenchymal cells) and are the least common type (about 1%) of human cancer. Sarcomas are a heterogeneous class of tumors and have several subclasses; osteosarcoma arises from bone tissue, chondrosarcoma arises from cartilage, liposarcoma from fat tissues (adipocytes), leiomyosarcoma from smooth muscle tissue, rhabdomyosarcoma from skeletal muscle tissue and angiosarcoma from blood or lymphatic vessel walls. Soft tissue is an open class of tissue which fills and connects the areas between the skeleton and the various organs

and the sarcoma subgroup soft tissue sarcoma (STS) are all sarcomas excluding those originating in bone.

2.2 PET

This chapter is inspired by a textbook on PET imaging (Phelps 2004).

Cancer diagnostics and PET

The most basic tumor grading is a measurement of the size of the tumor. As the tumor develops it generally grows in size, though this may not reflect its malignancy (rapid growth of the tumor on the other hand might). Most tumors only have a small fraction of stem cells, that is the true immortal cells that divide infinitely, the rest of the mass is differentiated and do not have clonogen capacity. Clonogen capacity is the ability to form new colonies called metastases; a defining feature of a cancer. This means that it is not necessarily the major mass of tumor that determines its malignancy but rather the number of aggressive cells. To measure the level of differentiation and make a prognosis a biopsy is required, but this is invasive and there is no guarantee that the sample is representative for the most aggressive part of the tumor. An alternative would be describing the gene expression through radiolabeled reporter genes, but this is limited by the complexity of the full biologic chain of events and the number of combinations of mutations producing similar results. The metabolism associated with proliferation and growth is on the other hand a common denominator and tracing glycolysis can visualize the activity. This tracing of glycolysis is the basis for FDG PET.

PET physics

PET, positron emission tomography, is an imaging modality where a molecule, used in a biological process in a target organ/cell, is labeled by a positron emitting isotope. A positron emitter is an isotope with too many protons compared to neutrons according to the valley of stability of nuclear physics. For large atomic numbers it might eject the proton directly, but for low atomic numbers it usually emits a positron instead. The positron is the anti particle of the electron, identical to it in most ways but with positive charge. The molecule labeled by the positron emitter needs to be of a kind that actively accumulates in the organ/cell and thereby visualizing its function. Figure 2 is a schematic of the chain of events leading to the visualization of the radioactive decay. When the radioactive isotope decays, a short range (ca

1mm in normal tissue) positron is emitted. The positron is slowed down to thermal speeds through Compton interactions and then merges with an electron annihilating them both. This converts their masses into energy in the form of two photons, both of 511keV. Due to conservation of momentum they are emitted back to back, though a slight deviation from 180˚ may occur as the electron/positron pair has some starting momentum. The PET machine is designed as a ring of detectors (scintillators) with an electrical circuit receiving time of detection and the energy of the interacting photons. If the energy of the photon deviates much from 511keV, usual range is 350-650keV, the detection is discarded as background noise. Two photons of correct energy detected within a few nano seconds of each other are assumed to have been created by an annihilation event along the line between the two detectors. The time difference, time of flight (TOF) information, gives a rough estimate where on the line the annihilation occurred and is stored together with detector positions. The list of these registrations are sorted and called a sinogram.

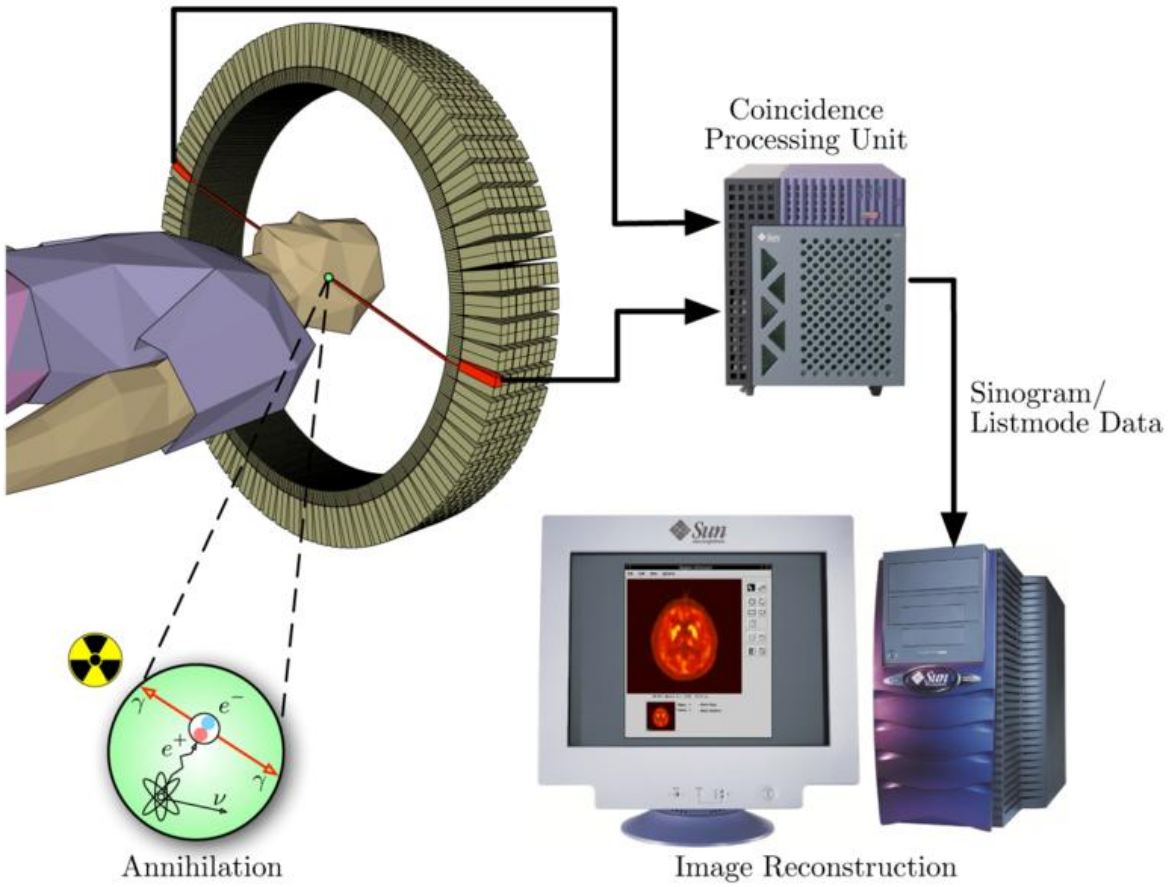

Figure 2. Schematic of a PET setup. The positron is emitted from the radionuclide, travels a short distance and annihilates with an electron. The energy is transported away as gamma rays interacting with a ring of detectors surrounding the patient. The two events are checked for coincidence and correct energy level and if appropriate the event is stored in the sinogram and used in the3D image reconstruction. (Wikipedia 2011)

Scanner corrections

To construct an image each sinogram entry is adjusted by a number of corrections. First, the pair of detectors constituting the sinogram entry is corrected for their orientation. The depth of the detectors, or rather the lack of depth detection, mean that there is a geometric uncertainty when the signal does not hit the detector head on. Second, the detector pairs are corrected for their detector efficiency. Each detector has an individual chance that the photons actually will deposit their full energy in the detectors and not pass through or get lost in a separator wall. Third, scatter inside the patient is corrected for two effects. On one hand the scattered photon represents a true event that is lost; this effect is corrected for by using the attenuation image created by the CT scanner integrated into the PET scanner. And on the other hand the deflected photon may not hit its correct detector but it can still hit another detector and register as a faulty detected coincidence. The energy window used to exclude faulty photons is limited by the detectors and it is large enough to accept photons that have undergone slight interactions. The chance of this occurring is calculated by simulation with the attenuation image or by using the discarded low intensity coincidences as a statistical measure of how common the event is. Random coincidences are impossible to distinguish from correct ones but their rate of occurrence can be approximated by the single event statistics or a dummy circuit that detects coincidences slightly delayed from each other. The value is dependent on the activity seen by the detector (collimation) as the true events are proportional to the activity while the random event ratio is proportional to the activity squared (there are two singles that need to interact). Another issue with high activity is that each time a detection event occurs there is a slight dead time, depending on the scintillation decay constant, before the detectors regain equilibrium. The frequency and duration of this effect is used to correct for lost detections within the time window. All these corrections are done internally in the scanner before any reconstruction is conducted.

Image reconstruction

For each entry in the sinogram the true position of the event is approximated by the event line. By summing all the event lines the true activity can be estimated. Since this in effect is an averaging along the line the resulting image is blurred and a sharpening filter should thus be applied. This process is called filtered back projection. The data is reconstructed in a matrix of small square boxes called voxels which represents the smallest objects we can visualize. Since the voxels are an average of all the activity within the box, objects need to be more than

twice the dimensions of the voxels before they start to be represented correctly. PET resolution is limited by a number of factors not present in CT or MRI. Though the detector crystal dimensions limitation is basically the same as in CT the energy levels in PET are generally higher and as such require deeper crystals amplifying the lack of depth determination and enhancing the geometric distortions mentioned previously. The positron itself wanders up to a millimeter and there is always some degree of annihilation photon acollinearity which is proportional to the diameter of the detector ring (1-2mm). And finally the reconstruction has a calculation precision where it is necessary to weight speed against precision. The resulting PET resolution is low compared to CT or MRI; usually it is about 5mm.

PET tracers

To avoid disturbing the true transport of the substance that is to be imaged, the marker isotopes are injected in trace amounts and are called tracers. There are several isotopes that are used in PET, and Table 1 lists some of them.

Nuclide	Half-life	E_{max}(MeV)	β^+ fraction	Use
^{11}C	20.4 min	0.96	1.00	Integrated into compounds used in the body, or molecules that bind to receptors.
^{13}N	9.97 min	1.20	1.00	
^{15}O	122 s	1.73	1.00	Oxygen uptake and distribution.
^{18}F	109.8 min	0.63	0.97	FDG, FMISO, FAc.
^{82}Rb	1.27 min	2.60, 3.38	0.97	Myocardial perfusion.
^{122}I	3.6 min	1.09	0.77	Thyroid metastases

Table 1. List of common nuclides used in PET. Half-life is the half-life of the isotope and E_{Max} is the energy of the emitted positron. β^+ fraction is the chance that the actual radioactive decay is a positron emission.

Two of ^{18}F's advantages can be seen. First, its half time is rather long so the time from cyclotron production to imaging time is not critical, while at the same time the halftime is short enough to prevent the patient from receiving unnecessary radiation. Second, the positron energy is low meaning it does not travel far before annihilation. ^{18}F can be used for hypoxia detection with FMISO, prostate cancer detection with FAc and fluorodeoxyglucose (FDG).

FDG is the most used PET assay, functions as a glucose analog tracer, and is designed to follow the same paths as glucose as it is transported through the body (Figure 3). Removing the second hydroxyl group from a glucose molecule gives a deoxyglucose molecule and replacing this hydroxyl group with an ^{18}F isotope gives fluorodeoxyglucose (FDG). FDG is transported in the bloodstream to the tissue and into the cells through their glut transporters. If the cells attempts to metabolize it through glycolysis the FDG will pass through the first step, phosphorylation by hexokinase (II), but since it no longer is compatible with the next enzyme it will be stuck in the process. Since the molecule now is charged it can not diffuse through the cell membrane and it is trapped inside the cell until the ^{18}F decays. At this point, the fluorine is transformed into an oxygen atom which in turn will capture a proton and revert into normal glucose and proceed through the rest of the metabolic cycle.

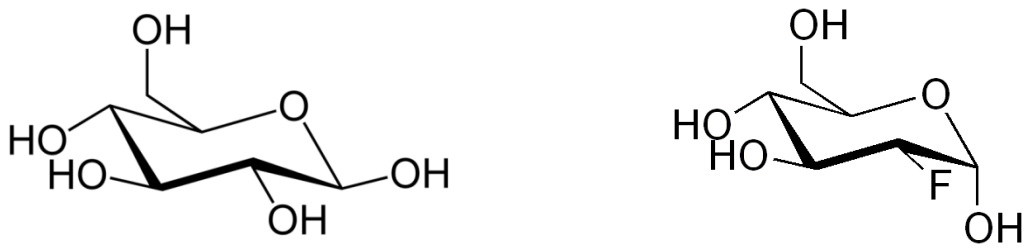

Figure 3. A cyclic form of glucose is shown to the left. Fluorodeoxyglucose is shown to the right.

FDG enter the cell by the same transport enzyme as glucose. In a normal FDG-PET scan the patient has fasted and blood sugar levels are low and the tumor starved. This means that membrane transporters are not saturated and changes stimulated through increased receptor activation is considered slow compared to our timeframe (less than an hour). Malignant tumors are often hypermetabolic (c.f. the Warburg effect discussed about) and are highlighted in the FDG-PET image.

Active smooth muscles like the intestines and myocardium also accumulates some FDG along with the nervous system which exclusively uses glucose as the source of energy. The kidneys filter the FDG from the blood and secret it into the urinary canals before storage in the bladder. And as such if imaging near the bladder is attempted a catheter is required to remove the noise. It is important to remember that detection is of the tracer and not the natural glucose and as such the values obtained is to a large extent dependent on the performance of the PET scanner and the reconstruction protocol used. This makes comparisons between different PET centers difficult, but standardization protocols have been published (Shankar, Hoffman et al. 2006).

SUV

To create a quantitative measure of tracer uptake that can be compared between patients, subjective elements needs to be corrected for. SUV, standardized uptake value, is a semi quantitative measure where the activity in a volume element is normalized to the injected activity and total volume over which the radioactivity is distributed, approximated by the mass.

$$SUV = \frac{radioactivity\ concentration\ \left[{}^{Bq}/_{kg}\right] \cdot Body\ mass\ [kg]}{Injected\ activity\ [Bq]} \qquad (2.1)$$

This is a common method used in the literature (Conrad, Morgan et al. 2004) though also criticized for its misuse. The SUV assumes homogenous absorption in the body which is not strictly correct as fat has a different absorption rate than muscle. This adds a gender bias and during treatment the common weight loss is primarily in fat leading to a shift as the treatment progresses. The most common alternatives either use patient surface or lean mass which corrects for the percentage of fat (Which is in most cases is a correction calculated from the height and sex of the patient). Normalization by mass (eq. 2.1) is however still the standard method and it has been reported in the literature that correcting for patient height, sex and glucose level does not translate into a detectable clinical benefit (Stahl, Ott et al. 2004).

The most common way of using FDG PET in cancer diagnostics is extracting the highest valued voxel element from an image acquired some time after injection when most tumors are still in an uptake phase, usually about 60 minutes after injection. It has been reported however that some tumors can accumulate as long as 5 hours after injection and that this time dependency discourages meta analysis and comparisons, limiting its use as a quantitative measure of malignancy(Keyes 1995). Meta analysis have been preformed however, and though there are few comparable parameters and the methodological qualities are generally poor it is indicated that the SUV value is capable of discrimination between sarcomas and benign tumors (Bastiaannet, Groen et al. 2004).

2.3 The two compartment pharmacokinetic model

The two compartment model is described in the literature and this chapter is inspired by a textbook on PET (Carson 2005).

2.3.1 Choice of compartments

A static image of the SUV value one hour after injection gives a good indication of the tracer uptake in the tumor. It does not however consider the dynamic properties of the accumulation. And as noted earlier in this chapter, the vascular structure and its leakiness are relevant malignancy markers of a tumor. To account for this, dynamic uptake data may be fitted to a mathematical model that represents the relationship between the measurable data and biological parameters. The concentration of FDG is an amalgam of a number of factors such as blood flow, plasma protein binding, blood percentage (hematocrit) capillary permeability, membrane transporters, tissue binding, receptor concentration, receptor association rate/duration, and metabolism in blood tissue and tumor. To quantify the problem a two compartment model has been described in the literature (Kamasak, Bouman et al. 2005).

The central limiting processes are considered the phosphorylation of the FDG by hexokinase, in effect trapping the FDG in the cell, and perfusion flow from the capillaries. These two processes separate the FDG into three compartments. First, one compartment contains the tracer ready for phosphorylation, which is partially in the interstitial space but also includes free FDG inside the cells. Second, one compartment contains tracer already phosphorylated and trapped inside the cell. Finally, one compartment contains the tracer in the blood plasma, the input value, and it is considered a known value not counted as an explicit compartment.

The rather coarse division makes the model robust and general without requiring full understanding of the entire underlying chain of events. Each compartment part is considered homogenous and the internal distribution is assumed to be much faster than the transport between compartments. In the blood there is assumed to be no difference between the central and distal part of the blood vessel. And the blood flow is assumed to be large enough that the decrease in glucose concentration along an artery is negligible. From the first compartment free FDG is transported into the second compartment by hexokinase. Hexokinase is only found inside the cells, usually bound to mitochondria. This means that cell membrane transporters need to be fast enough that the concentration of free FDG is the same within the

cells as in the interstitial space, otherwise the barrier between the compartments become an amalgam of hexokinase and glut transporter expression. FDG is however preferred over glucose by the membrane transporters, while hexokinase prefer glucose. The transport rates are considered constant within an hour, biological processes are concentration dependent but changes are assumed to be slow. Another issue is that the measurement is not of the substance itself but of the tracer. However, it has been shown that as long as the tracer concentration is low compared to the substance concentration we have a linear enzyme-catalyzed reaction.

2.3.2 Compartment exchange rates; the K values

The parameters that represent the rate of transfer of a substance are called k, much like heat is radiated dependent on the temperature of an object. The amount of substance transferred from compartment A to compartment B is labeled J_{AB}. The concentration of substance in compartment A is labeled C_A. The transfer factor governing the transfer from A to B is labeled k_{AB} and is defined to fulfill the equation:

$$J_{AB} = k_{AB} C_A \tag{2.2}$$

This is a linear relationship assuming that the cellular (glut) transporters are unsaturated and that the capillary surface is large enough to make the perfusion linear to flow. In the case of the two compartment model the subscripts are simplified into K1, k2, k3 and k4 (Figure 4), K1 is in capital to denote it is related to flow while the others are related to concentrations.

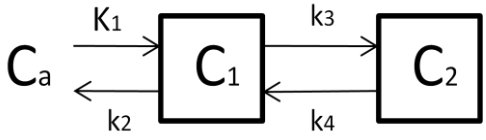

Figure 4. Schematic of the two compartment model. C_a is the plasma concentration, $C_{1\ is}$ the free (interstitial) concentration, C_2 is the bound (metabolized) concentration and K1, k2, k3, k4 are the transfer rates between compartments.

The rate of change of concentration in the free compartment (first compartment labeled C_1) is the transfer from the plasma ($C_a \cdot K_1$) into the free compartment minus the transfer from the free compartment into the plasma ($C_1 \cdot k_2$) minus the transfer from the free compartment into the bound compartment (second compartment labeled C_2) inside the cells ($C_1 \cdot k_3$).

$$\frac{dC_1}{dt} = K_1\, C_a(t) - k_2 C_1(t) - k_3 C_1(t) \tag{2.3}$$

In the general case there should also be a k_4 component with transport from the second to the first compartment, but since the rate of dephosphorylation of FDG is slow and the charged molecule is incapable of crossing the cell membrane k_4 is approximately zero. The rate of change of concentration in the bound (phosphorylated) state is the transfer from the free compartment into the bound compartment:

$$\frac{dC_2}{dt} = k_3 C_1(t) \tag{2.4}$$

Assuming a perfect bolus input function, that is a Dirac pulse with the amplitude C_a, the solution of the differential equations becomes:

$$C_1(t) = C_a K_1 e^{-(k_2+k_3)t} \tag{2.5}$$

$$C_2(t) = C_a \frac{K_1 k_3}{k_2 + k_3}\left(1 - e^{-(k_2+k_3)t}\right) \tag{2.6}$$

The tissue activity is linearly proportional to the magnitude of the input and the superposition principle can be used to create the solution for a series of boluses at times T_i with the input magnitude $C_a(T_i)$:

$$\mathrm{C_1(t)} \;=\; \sum_i C_a\,(T_i) K_1 e^{[-(k_2+k_3)(t-T_i)]} \tag{2.7}$$

$$C_2(t) = \sum_i C_a(T_i) \frac{K_1 k_3}{k_2 + k_3}\left(1 - e^{-(k_2+k_3)(t-T_i)}\right) \tag{2.8}$$

A continuous input function can be approximated by approaching an infinite summation of boluses. This gives an integral that can be identified as a convolution:

$$C_1(t) = \; C_a(t) \otimes \left[K_1 e^{-(k_2+k_3)t}\right] \tag{2.9}$$

$$C_2(t) = C_a(t) \otimes \frac{K_1 k_3}{k_2 + k_3}\left(1 - e^{-(k_2+k_3)t}\right) \tag{2.10}$$

The total tissue tracer concentration is the sum of the two and when measuring the values of the computational elements (voxels) the two compartments cannot be distinguished:

$$C(t) = C_1(t) + C_2(t) = C_a(t) \otimes \left(\frac{K_1 k_2}{k_2 + k_3} e^{[-(k_2+k_3)t]} + \frac{K_1 k_3}{k_2 + k_3}\right) \tag{2.11}$$

This equation describes the ratio of the input function (plasma) to the tissue function. The volume inside the arteries is a composite of plasma and red blood cells called whole blood. With an invasive blood sampling of the artery, or with some modifications from a vein, the blood can be centrifuged to separate the plasma. PET imaging on the other hand gives the average of the two while it is only the plasma that is transmitted over the capillary walls. The metabolism in the red blood cells is low however and the exchange rate between the plasma and red blood cells are fast. Furthermore the permeability surface is assumed to be large enough to not constitute a restricting factor. The K_1 factor is then the linear link between blood flow and FDG transfer rate.

2.3.3 Patlak

Expressing the convolution as an integral gives:

$$\begin{aligned} C(t) &= \int_0^t C_a(s) \left(\frac{K_1 k_2}{k_2 + k_3} e^{[-(k_2+k_3)(s-t)]} + \frac{K_1 k_3}{k_2 + k_3}\right) dt \\ &= \int_0^t C_a(s) \frac{K_1 k_2}{k_2 + k_3} e^{-(k_2+k_3)(s-t)} + \int_0^t C_a(s) \frac{K_1 k_3}{k_2 + k_3} ds \end{aligned} \tag{2.12}$$

Assuming the input function to be constant and t long enough so that the exponential becomes sufficiently small, a linear equation is obtained:

$$\frac{C(t)}{C_a} = \frac{K_1 k_2}{(k_2 + k_3)^2}\left(1 - e^{-(k_2+k_3)t}\right) + \frac{K_1 k_3}{k_2 + k_3} t \approx \frac{K_1 k_2}{(k_2 + k_3)^2} + Kt \tag{2.13}$$

The slope of the equation, K, is the product of the entry rate into the tissue from the plasma, K_1, and the fraction of tracer in the tissue that reach the irreversible compartment two, $\frac{k_3}{k_2+k_3}$.

In the case the input function is not constant the Patlak transformation is:

$$\frac{C(t)}{C_a(t)} = V_0 + K\left(\frac{\int_0^t C_a(s)\, ds}{C_a(t)}\right) \tag{2.14}$$

Here V_0 is a constant, the initial volume of distribution. The volume of distribution represents the volume the tracer would occupy if it had the same concentration in tissue as in blood. A

while after injection an equilibrium can be imagined between the tracer delivered by the blood flow and the tracer leaking back. The concentration in compartment 1 becomes a constant along with the ratio C/Ca = K1/k2 which is called the volume of distribution, which is the relation between concentration in blood and tissue. By combining equations 2.2 and 2.3 the total transfer rate from blood to compartment 2 at equilibrium becomes$\frac{K_1 k_3}{k_2+k_3}$. This transfer value from blood to the phosphorylated state is a measure of the metabolic rate:

$$MR_{FDG} = \frac{K_1 k_3}{k_2 + k_3} \tag{2.15}$$

2.4 IDL

IDL, Interactive Data Language, is a computer programming language from 1977. The most basic form of a computer programming language is a list of commands that are executed after each other (script), originally stored on punched cards. It is natural to cluster commands that are always executed together and define generic sub routines called procedures that can be called directly from the main list of commands or from within the subroutines themselves. This is called procedural programming and makes the code easier to both implement and read. If the list of commands are replaced by an infinite loop (main loop) divided into one part detecting events, and another processing events it is called event-driven programming. The events are produced outside the control of the programmer and the exact order of the commands is not known when the program is created. This results in increased flexibility and complexity and is a requirement for most user interfaces. IDL is primarily a procedural programming language with focus on smaller programs though it contains some support for the more complex paradigms. (A paradigm is the manner the programmer organizes the procedures and data types)

IDL is mainly used within astronomy and medical imaging and has built in several imaging processing procedures; one such is support for reading DICOM files. DICOM, Digital Imaging and Communications in Medicine, is a standard for file exchange in medical imaging developed by NEMA, National Electrical Manufacturers Association. IDL is an array based programming language resembling FORTRAN. It is dynamically typed and the variables need not be pre declared making it flexible, but at the same time limits the errors the compiler is

capable of detecting. When the compiled program has finished executing, new commands can be entered interactively with the command prompt while still having access to the global variables. This makes it easy to make small adjustments but it is still limited in more complex structures, especially since IDL has only one namespace.

IDL supports graphical user interfaces, GUI, through the use of widgets which are event-driven. It also has some features for basic object oriented programming. Object oriented programming lends its concept from nature where all organisms are composed of small simple self sustaining elements, namely cells. Though all cells have the same parent they differentiate to solve different tasks, but they always inherit certain traits. To facilitate the construction of large complex programs a similar paradigm was created. By encapsulating data types and functions used by an element and only providing an external interface to that element chunks of the code should be made self sustained and replaceable in case of an upgrade or a new programmer using the code.

3 Methods and materials

3.1 The graphical user interface (GUI)

Since the operator of the program needs to see the PET images and to define areas inside a volume it was decided to decided to create a GUI, graphical user interface, where it is possible to dynamically add and remove volume elements. To adjust the calculation parameters a preference menu is made along with a file format (.lva) for saving the settings. Two windows are used for displaying the results and a file format (.trc) is used to save the results (output data). Finally a second analyzer program is created that read the .trc data files and create scatter plots and data set comparisons. The structure of the first program is displayed in Figure 5 and the structure is similar in the analyzer program. It is divided into two primary parts. The first part, the front end, deals with the GUI and the location and values of all the sliders and buttons. This is the lguiData object and it receives events from the xmanager which are processed and transmitted to the second part, the back end. In the back end all calculations are conducted and the results are transmitted to the front end for display or to the file IO structure for saving/loading.

The program is designed with an observer structure. The event handler (xmanager) is separated from the GUI setup which is in turn separated from the calculation algorithm. The event handler is an infinite process that lays dormant waiting for some activity from the operating system (OS). When an activity like a mouse movement or key click is detected it reports this event through a set of callback functions called with a shared data set, lguiData, as a parameter. This constitutes the interface between the GUI layout and the xmanager. The buttons defined all exists within a context hierarchy with a top level base and the interface, lguiDataPtr, is supplied to this structure (Where ptr signifies that it is a pointer to the memory segment). The lguiDataPtr is essential as the callback functions themselves does not have access to the memory segment of the main function and this is their link to it. The bases define the positions and internal relationship between the elements various sub bases. The code defining and creating the GUI is found in Appendix D: GUI definition.

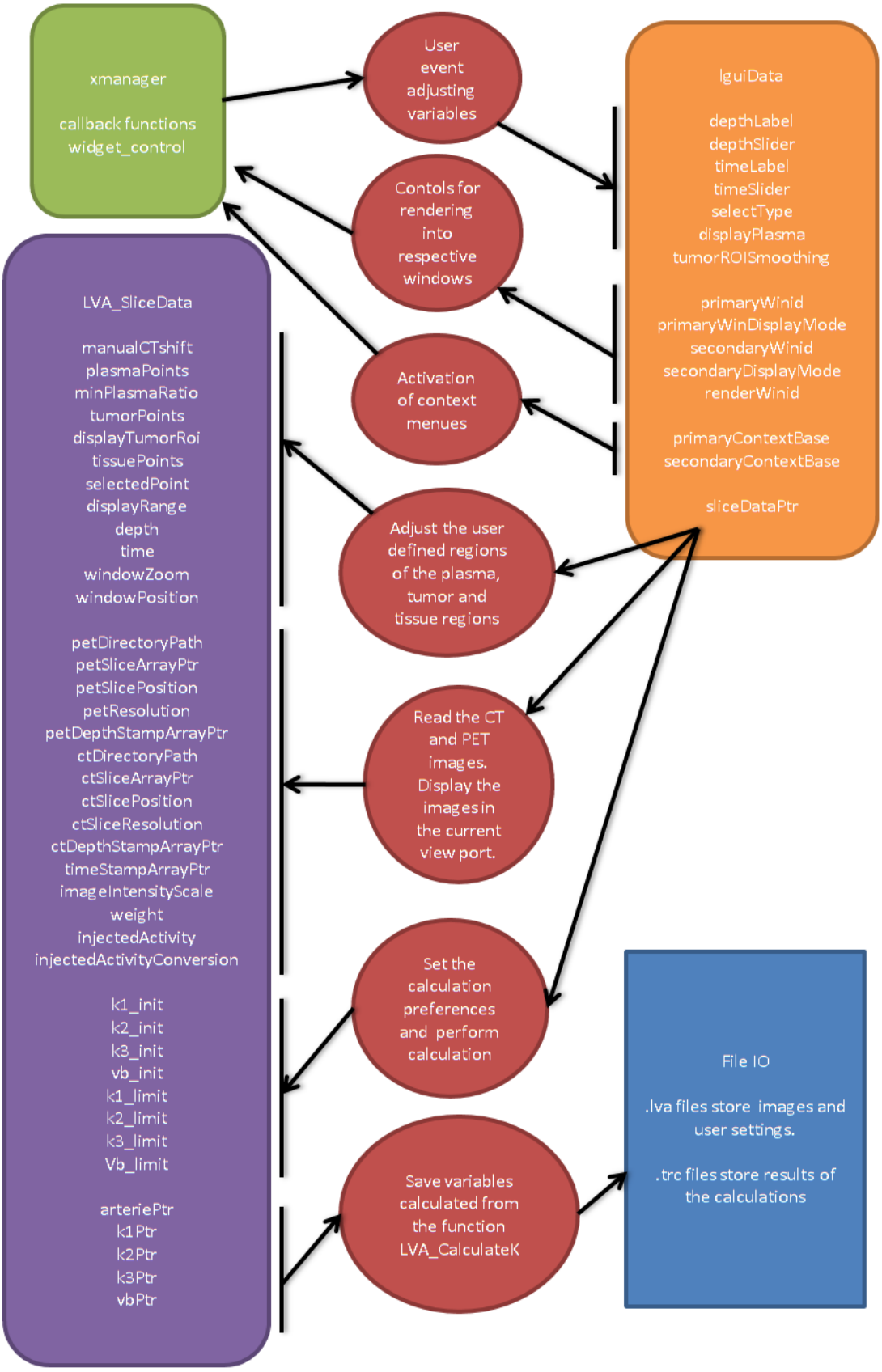

Figure 5. The interface structure and corresponding data structures. Arrows indicate and interface interaction, boxes indicate data objects and ellipses indicate actions.

The lguiData works as the interface between the various GUI elements and contains 4 main types of data. First, the variables that saves the state of the sliders and buttons. Second, the windows into which we render our images. Third, the context handlers containing the hierarchies of buttons contained within menus that appear only within a certain setting. (Pressing the right button over the left window will result in a different menu than if over the right window or pressing the button over a submenu created by an earlier mouse click). Fourth, it contains the pointer to the LVA_sliceData, which is the interface to the calculation algorithm. The LVA_sliceData contain the data and functions for manipulating and displaying the original PET/CT images and the resulting images from the calculations. (Appendix D: Definition of the LVA_sliceData object).

The LVA_sliceData structure contains 3 types of data. First, the raw image data loaded from the DICOM files. Second, the user defined variables set from the GUI. Third, the data calculated from the pharmacokinetic model. The interface contains functions for adjusting and reading the variables (get/set functions), display functions for drawing / plotting the images, IO (input / output) functions for writing or reading the data from the hard-disk and calculation functions for plasma fitting, k values and Patlak values. Since the calculation data is encapsulated it is not dependent on the UI and allows scripting of the event structure for automatic processing removing the necessity for repetitive UI operations. (Appendix D: Calculation functions)

As so many operations depend on comparisons there are two viewports that can be set independently. The general layout of the program is seen in Figure 6, in this instance the PET frame and CT frame has been displayed side by side. In the left viewport the user is in the process of marking the tumor, seen as a lump both in the PET and CT images. To define a tumor region the correct slice is selected by the depth slider and from the region slider “Select tumor” is chosen. A right mouse click now triggers an event from the xmanager, the lgui receives and sorts this event deciding to open the dropdown menu and inform the xmanager of the new context menu. When the next mouse event occurs within the menu the xmanager triggers the lgui callback function of the “Define ROI” button which in turn creates a new window for selecting primitives (Figure 7). The user now traces the corners of a polygon with the mouse and this area transmitted trough the lguiData interface down into the LVA_sliceData object. This operation is repeated for all slice depths until a full tumor definition is obtained.

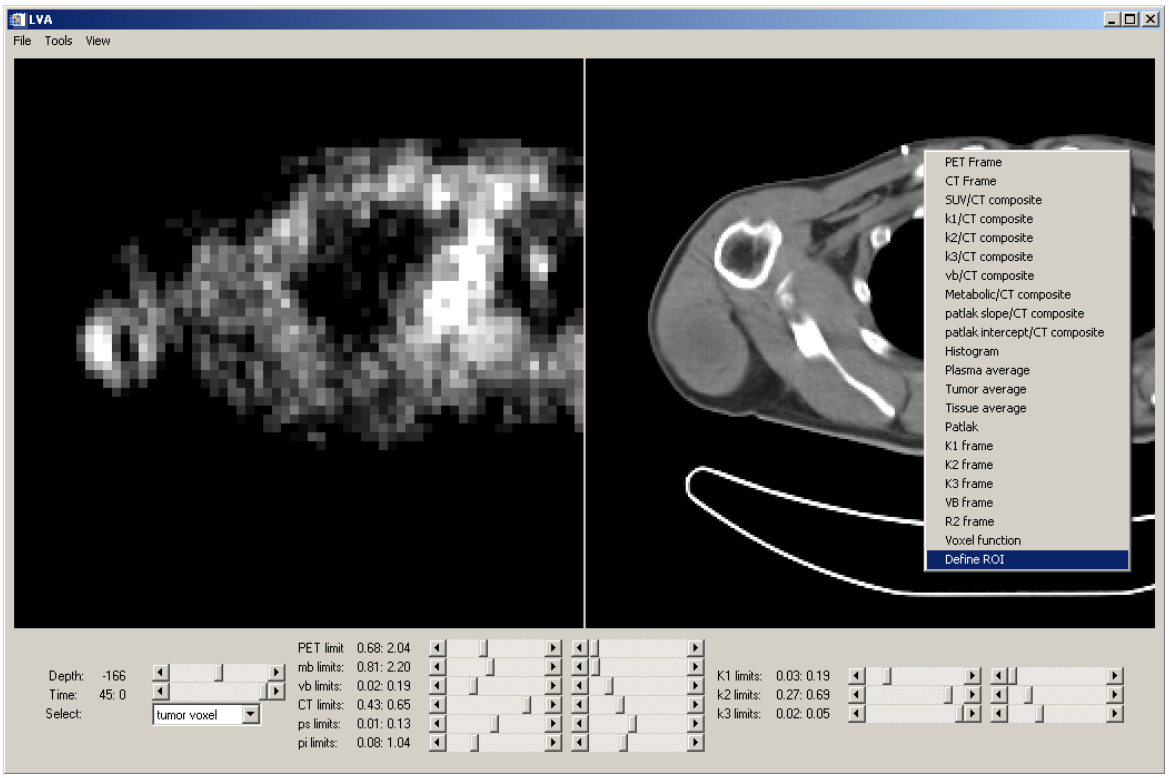

Figure 6. Image of the program. The depth and time slider adjusts the depth and time of the displayed slice. The select button chooses what type of region is defined. The limit sliders set the thresholds of the output display. The dropdown over the right viewport chooses what should be displayed in the viewport.

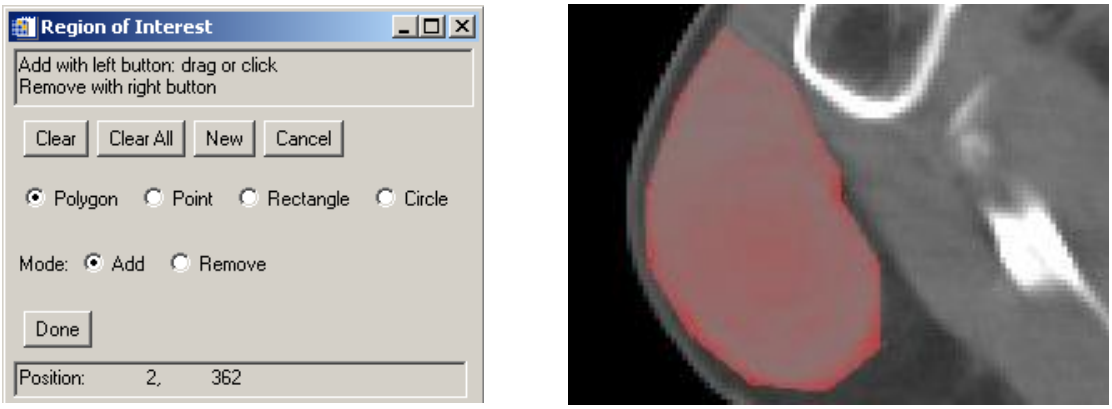

Figure 7. The image to the left is the menu for selecting regions of interest. To the right a tumor selection is seen as a red polygon with 12 corners.

3.2 PET/CT scanning and data preprocessing

The patients selected for the current study all have large soft tissue sarcoma of grade III or IV. Since the dynamic PET protocol is experimental, written informed consent is obtained from the patients. The study was approved by the regional committee for research ethics. The current work has not influenced the choice of treatment planning for the selected patients the

only modification is that additional images are obtained. The final static images are still provided as in a regular examination. In practical terms it means that the patient needs to remain in the scanner for 60 minutes rather than 15 minutes.

Seven of the eleven patients included underwent surgery soon after the PET scan, and only one scan per patient was thus in this case obtained. Four patients were subjected to radiation therapy (RT) and scans where undertaken at different time points through the treatment, one during the first week, one during the last week and finally one 3 months later. Before treatment the patients are scanned and tumor definitions are entered into the radiotherapy planning program Masterplan. The gross tumor volume (GTV) defined in Masterplan is used for delineation of tumors in patients subjected to RT.

Before the examination the patients are told to fast, that is to not eat prior to the examination (min 6 hours). This is done to lower the glucose level in the blood thus letting the body switch to fatty acids as the main source of energy. The image is then less contaminated from the glucose activity in the muscles and the low glucose level limits the risk of transporter saturation in the tumor as it is glucose starved. The patient is kept warm and comfortable during the examination as anxiety and shivering lead to FDG uptake in muscles. Good hydration is required for proper FDG secretion and the patient is made to drink water before the examination. The scanner used for all patients is a Siemens AG Biograph 16. The radiographer acquires one CT image at the start of the examination and a sequence of PET series with 15s intervals the first 3 minutes, 30 seconds intervals the following 8 minutes and finally 2 minutes intervals up until the end at 45 min. Attenuation corrected PET images are then generated by the acquired sinograms in the PET scanner and saved to the DICOM file format. The glucose levels of the patients are obtained and found to be in the range 5.1-6.8 mMol which is considered reasonable.

The DICOM files are read in IDL using a built in library. The reader is encapsulated in the object IDLffDICOM and its use exemplifies the IDL object structure. The functions used to access and modify the objects values, in this case Read() and GetValue(), are the interface. The internal variables themselves are not used directly and in that way they are protected.

```
myDicom = OBJ_NEW('IDLffDICOM')
read = myDicom->Read(filename)
dimXYptr = myDicom->GetValue('0028'x, '0010'x, /NO_COPY)
…
OBJ_DESTROY, myDicom
```

The OBJ_NEW allocates the memory needed to store the data read from the file, and the OBJ_DESTROY frees it. The two values used in the GetValue() function call are two hex values, the x in `'0028'x` denotes that it should be read as a hex number, that represents the code for each variable extracted from the data object. The stored attributes and their corresponding hex codes are defined in the DICOM header and read by the program efDcmDmp (Table 2). Each image has in addition to the image data a spatial position (offset from the isocenter), x/y/z resolutions as well as a time stamp. The rows and columns are 128 with a pixel spacing of 5 mm (standard PET resolution) for all patients. The slice thickness is 2 mm for all patients while the slice spacing, as seen by the image position, is variable. The first sequence of patient 2 has a 2mm spacing while sequence 2, 3, 4 of patient 10 and the entire sequence of patient 11 has 3 mm spacing. All other patients have a spacing of 4mm. The position data is also used to synchronize the isocenters of the PET and CT scans. Rescale slope is a conversion factor to map the values stored in the data array (chosen for precision) to the actual units, houndsfield units in CT and MBq/ml in PET. The instance number and acquisition time is used for sorting the sequences.

Hex codes		Dicom value
0028	0010	Rows
0028	0011	Columns
0028	0030	Pixel spacing
0018	0050	Slice thickness
0028	1053	Rescale slope
0020	0032	Image position
0010	0030	Patient weight
0018	1074	Radionuclide total dose
0020	0013	Instance number
0008	0032	Acquisition time
07FE	0010	Pixel data

Table 2. The list Dicom hex codes used by the DICOM reader. The two leftmost columns are the identification tag. The Dicom value is the value returned by the IDLffDICOM object when the GetValue(hex1, hex2) is called.

Also included in the image data are the weight and injected activity, which are used to convert the data into a SUV map. The injected activity is sometimes represented in pC (pico Curie; 1Ci = 3.7×10^{10} Bq) instead of Bq. However, the DICOM format does not contain information on these units, and a conversion was made based on the magnitude of the injected activity. Since the PET and CT images do not always overlap precisely using the positional data the CT position is adjusted and corrected for using the variable ‘ manualCTshift’ (Figure 5). To

determine this value visually the PET signal is displayed in transparent red overlapping the CT image (alpha blending). The operator search for some correlation of the PET signal with an identifiable structure on the CT image, such as bladder or arteries and try to get a good match. This adjustment is minor and rather subjective and only affects the borders of the tumor when the CT image is used for delineation. Note also that the true resolution of the PET image is much lower than for the CT and in calculations the true resolution is always used while when the image is displayed a smooth image is interpolated on the CT image (Figure 8). The size of the box pattern clearly reveals that the PET voxels are an average over regions of more than one type of tissue.

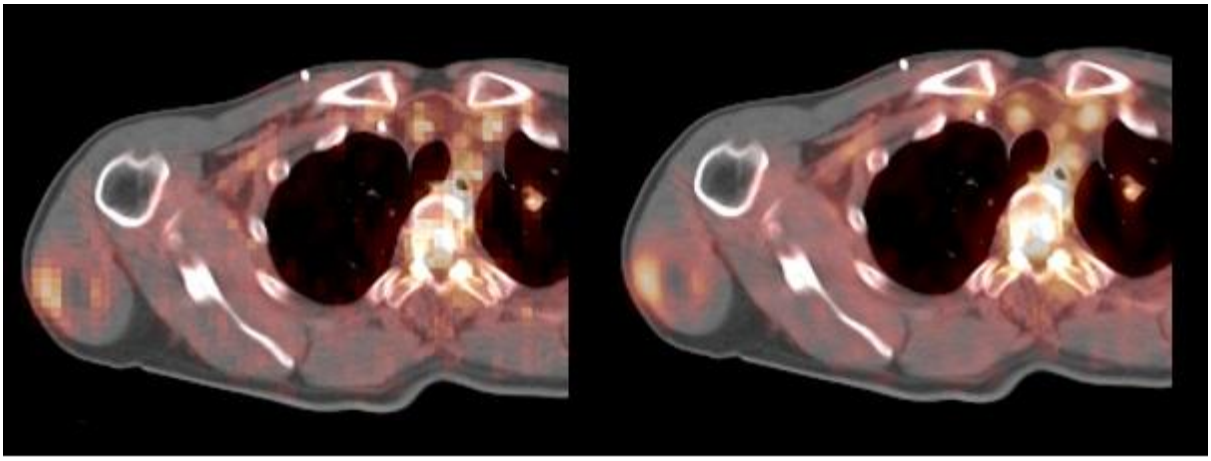

Figure 8. PET composite image of patient 10. The left image is a CT slice alpha blended with a PET slice. The right image is the same CT slice alpha blended with the PET slice interpolated to the same resolution as the CT slice.

3.3 Regions of interest (ROIs)

The input into the mathematical model described in chapter 2.4 is the plasma function. To estimate the plasma function non invasively we use the dynamic PET images directly, a process described in the literature (Ohtake, Kosaka et al. 1991). The injected radiotracer is initially transported in the blood stream. And as such the arteries are seen at the first PET images as the first regions to show activity. Since the tracer needs time to distribute in the entire vascular system and the injection time is not known precisely a single image is not used for identification of the vasculature. Instead we sum the images of the first 45 seconds, when the FDG is still contained to the arteries. From this PET image the arteries are identified and then the position is verified with the CT image where the vessel walls can be seen. Switching between the two modes allows them to supplement each other. The CT frame rules out

regions of noise like the region around the heart while the PET frame corrects for motion of the aortas, as the CT image is a static image acquired at the beginning of the examination.

Due to partial volume effects, voxels are selected from the large vessels like the arch of aorta or thoracic aorta when possible. Selecting central points as sample points for the plasma function avoid the influence of the vessel walls. The heart (left ventricular chamber and atrium) is a good source of blood but it also contains large muscles (myocardium) and much movement complicating the sampling. The aorta is therefore preferred if it supplies sufficient sample points. Figure 9 shows an example where the ascending and descending aorta is sampled from right below the arch of aorta (See Appendix: A for anatomical terms). Usually the peak appears 30 to 45 seconds after injection but sometimes the scan is not properly synchronized with the injection and the peak appears on the first image. In this case it is not guaranteed that the peak has been reached and amplitude and shape may be slightly truncated.

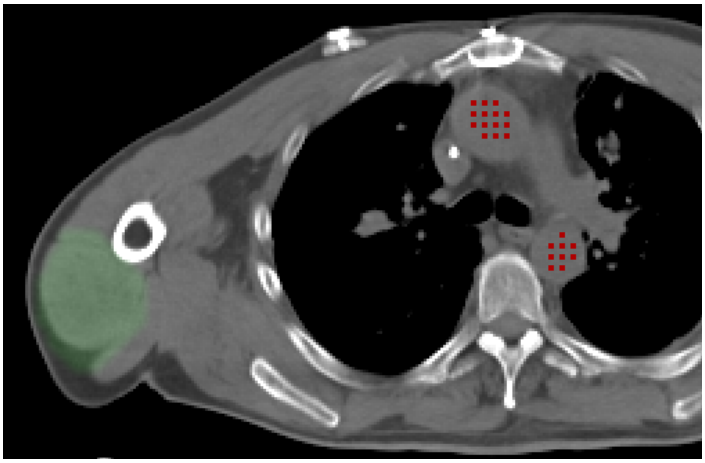

Figure 9. Visualization of the selection of the aorta of patient 10, the red squares represents the plasma sample points. The green region is the tumor.

Tumor regions are delineated either in the CT image where they can be identified as solid masses, or as hyperintense areas in the final PET images. In the latter, FDG has accumulated in the tumor but areas such as the kidneys, bladder and nervous system do the same and can lower the image contrast if located close to the tumor. As PET and CT complement each other a combination is usually used by adding the CT frame to one viewport while using a PET frame in the other viewport. The CT image has the highest resolution and the polygon created to encompass the tumor is created in this resolution. However the PET resolution is used in all calculations and the CT-based polygon is downsampled to this resolution. If the downsampling distorted the intended shape the user can add or remove voxels manually by

marking them after the region is selected (Figure 10). One such adjustment is using early frames to identify arteries passing through the tumor and excluding the points.

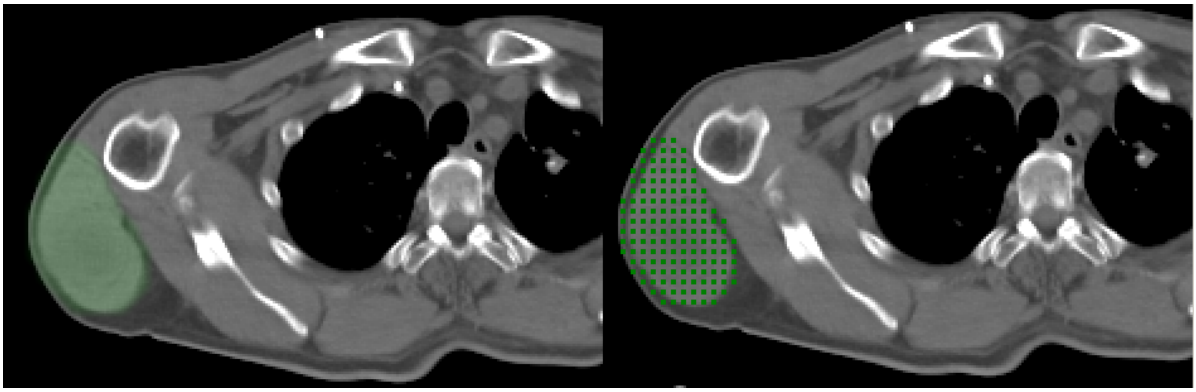

Figure 10. CT scan of patient 10. To the left the tumor definition is marked in green. To the right the same tumor is shown with green rectangles representing the correct PET voxel resolution. Marking one of the green rectangles adds or deletes voxels from the ROI.

It is inevitable that there will be some systematic errors it the scanner setup as no two scanners are exactly the same, however the systematic errors inherent in our imaging technique should share a common variance in two comparable regions in the image. By calculating the ratio between the regions certain methodological errors may possibly be eliminated giving a value that is independent of the particular scanner. In addition there is a certain geometric distortion for all detector pairs depending on if their line of response, LOR, passes close to or far from the center (Figure 11). By choosing the reference regions at the same distance from the isocenter this effect may be reduced. Calculating the ratio between the tumor values and a reference tissue has been described in the literature as a way to remove scanner bias, however, it is also found to introduce additional interobserver variability as a second region also need to be defined (Benz, Evilevitch et al. 2008). Since the results have been inconclusive the tumor-to-tissue ratio is chosen as one of the parameters tested in this thesis and is referred to as the SUV_r. As reference tissue we use muscle, preferably at a mirrored (contralateral) site in the body. For instance if a tumor is situated near/in the left deltoid, sample tissue is delineated in the right deltoid. This way the tissue though not the same as in the tumor, has the same geometric offset from the scanner isocenter and represents the blood supply the tumor region would have had if it was normal tissue. Circulation will of course never be exactly the same in the two regions but it allows selection of similar setting while avoiding possible contamination resulting from the proximity to the tumor if reference tissue was chosen right outside the border of the tumor.

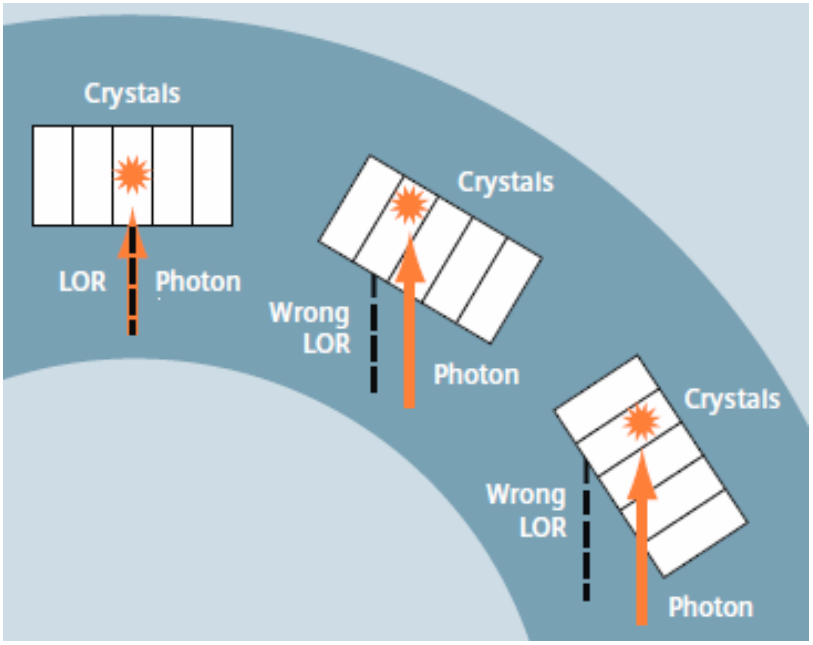

Figure 11. Illustration of the effect of geometric distortion in a PET scanner. From left to right, photon interactions in the detector crystal head on (isocenter) to interactions from points in the periphery. (Siemens 2009)

3.4 Tumor descriptors

The output from the dynamic PET analyzer is on a voxel basis and come in a 4D or 3D matrix, and the results are presented as images or condensed into histograms that are described by percentile values and central moments such as variance, skew and kurtosis.

A histogram is a list of how many data points that are confined to a certain range of values. The total value range, that is the maximum value of the data points minus the minimum value of the data points, is divided into equal sections called bins. Usually there are 3-100 bins, larger bins reduce noise and enhances robustness while smaller bins gives higher resolution. Then, for each bin, the number of voxels in the tumor that have a value within the bin range is counted. This value is then normalized to the total number of tumor voxels. For the 4D arrays this process is repeated for every time point yielding an array of histograms.

The histograms are in the current work described by two methods, percentiles and central moments. The x^{th} percentile of a histogram is the value which x percent of the values in the histogram are less than or equal to. Central moments are used to describe probability

distributions and four orders of them are used in this thesis. The first order is the mean, which is the ‘center of gravity’ of the histogram. The second order is the variance which describes how far the values are spread from the mean. The third order is the skew which describes to what extent the spread of points lie to the right (positive), to the left (negative) or are evenly distributed (zero). The fourth order is the kurtosis which describes if the spread is centered on the mean (low) or out in the edges (high).

Reducing the values to the mean includes all data points but gives a dependency on the definition of the border and as such introduces inter observer variability. If the delineation of the tumor is done in a strict way less of the surrounding tissue is included and a higher mean will be calculated. The value is however robust and rather independent of noise. At the other extreme the tumor could be represented with its single highest value, in this text it is referred to as ‘Max’. The malignancy of the tumor is usually characterized by the most active areas of the tumor, and the Max value represents the most active volume element. This is the most common descriptor of standard PET images. It is however a single point and as such prone to noise. Extending this definition by including the six surrounding voxels(star shape), in effect increasing the size of the voxel element, and taking the mean of this region give a value labeled ‘Peak’ in this text.

3.5 Data analysis

Normalizing the recorded values into a SUV image (equation 2.1) of the tumor is the most basic and most used descriptor of the tumor. The reference tissues used to calculate the SUV_r are from regions with low activity compared to the tumor and are more susceptible to random noise. The value representing the reference tissue is the mean value of the region smoothed with a simple 3 average filter in the time dimension. The percentiles and central moments of the SUV and SUV_r constitute the information obtained from a static PET image. Two superscripts are defined for early and late SUV values. A window of 1:30-2:30 min is used for sampling of what is hereafter labeled as ‘early’ SUV; SUV^E. This averaged image will later be used to represent a simple measure of vascularization/perfusion. The sampling interval is chosen to be as early as possible as this is when the bolus is in its most concentrated form, yet not to early as there are uncertainties in the actual injection time and latency before the bolus reach the tumor. Metabolic accumulation of FDG continues through the entire PET scan and when choosing an interval describing it later is better. The interval 41:00-45:00 is labeled

'late' SUV; SUV^L. This averaged image will later be used as a simple representation of the metabolic activity.

Partial volume effects

To calculate the dynamic properties the plasma function is required (Chapter 2.3). Generally 20-400 plasma points are defined depending on if the scan is of the legs or the torso. In the legs the limited spatial resolution of the PET scan and the diameter of the arteries introduce partial volume effects. Consider an object with the dimension of a voxel: only if the object precisely coincides with the voxel will a correct value will be obtained. Most likely the object will be shared between voxels averaging the value with the surrounding tissue. The artery, being approximately a cylinder, should have a diameter 2-3 times the size of the voxel to represent the true activity in it. The arteries, especially the smaller ones in the legs, will in some slices display a fine peak while the artery is centered on the voxel grid. A slice lower there will be instead two connected peaks of fractions of the original value as the artery has shifted slightly in a direction and is now partly represented in the voxel with the old center point and partly in a new voxel (See Figure 12). At some deeper slice the artery is refocused on the voxel grid and a full value is again obtained. This effect makes an automatic tracing of the artery through 'region growing' more complex and also creates points in the artery where the true activity in artery cannot be represented by a single voxel.

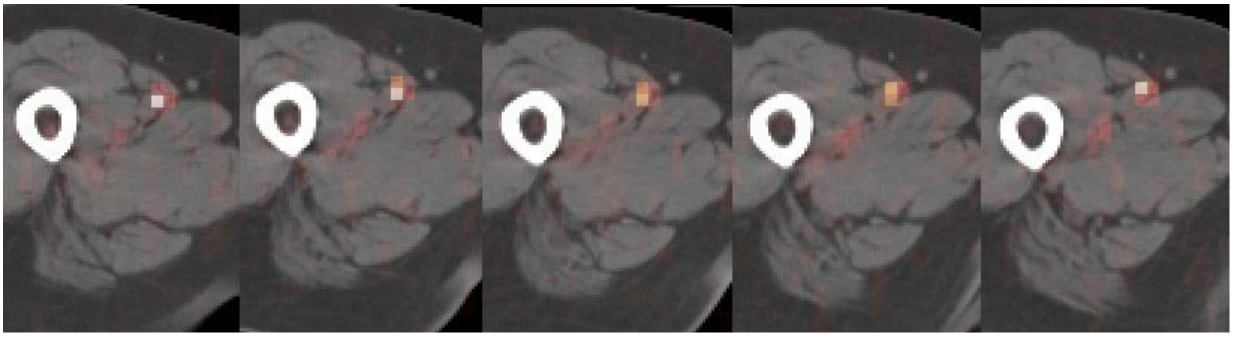

Figure 12. The partial volume effect on a femoral artery of patient 1. From left to right, each consecutive axial image plane is shifted 4mm caudally.

To remove the voxel values clearly influenced by partial volume effects, the program calculates the quality of the points. The initial values are compared to the late values and when the ratio between the two decreases below a certain threshold the point is disregarded. This way only the voxels centered correctly with the artery are used. The mean value of the plasma points is fitted to a bi exponential function (ignoring the points prior to the peak). Since the sample times are unevenly distributed a full time array with equidistant step is

created and the corresponding value of the bi exponential is calculated. The raw values prior to the peak are copied and the rest of the points are calculated from the bi exponential. The resulting array is used as the input function.

Patlak analysis

Examining only the SUV values at the end of the PET scan disregards the time trend of the FDG accumulation and implies a dependency on the timing of the final scan. A Patlak plot is a graphical way of describing the dynamic development of the PET images utilizing the plasma function (Chapter 2.3.3). The actual signal from the tumor (total accumulated) normalized to the tracer concentration in the plasma is plotted against the integral of the circulation (total delivered) normalized to the tracer concentration in the plasma (Figure 13).

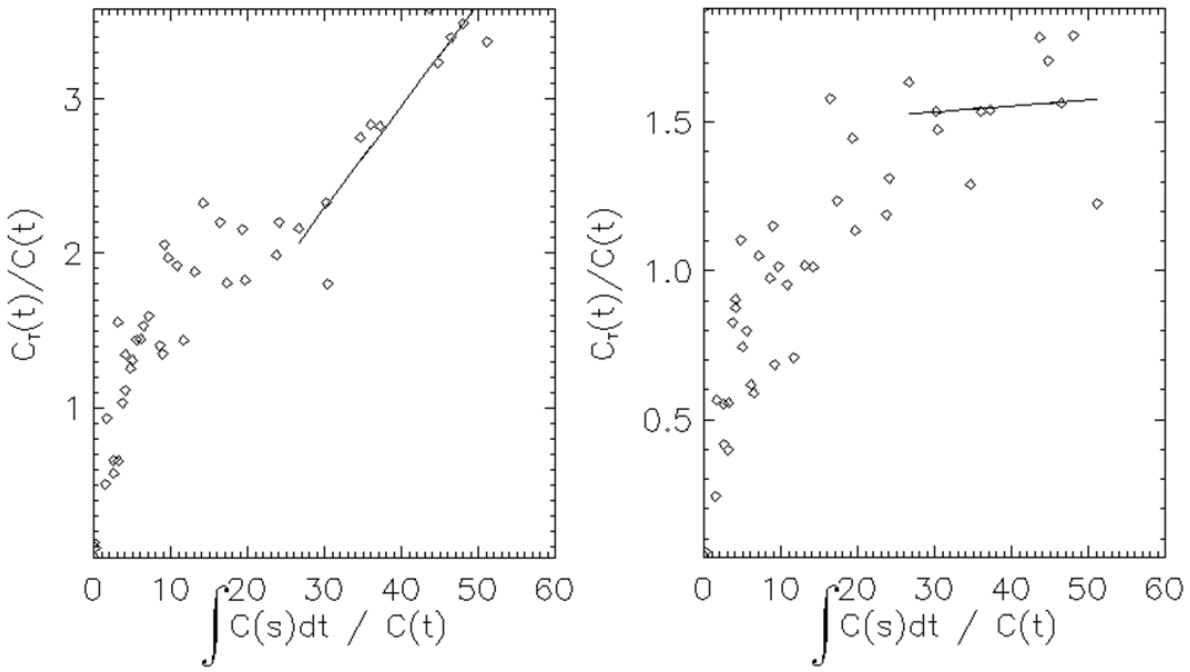

Figure 13. A Patlak plot from patient 1. The left plot is from a central tumor point and the right plot is from the tumor edge. On the ordinate the voxel activity [$C_T(t)$] is normalized to the plasma activity [C(t)]. On the abscissa the integral of the plasma activity up to the point t is normalized against the plasma activity. The solid line visualizes the result of the linear regression.

Some time after injection a linear relationship may be observed and a linear regression can be performed. The slope and intercept from the regression line is extracted. The slope of the curve represents the rate the tumor is metabolizing the FDG from the surrounding bloodstream. The intercept represents the tissue concentration of FDG resulting from the initial bolus. The resulting images are displayed either alone or composited with the CT

frame. The images are also reduced to histograms and represented as percentiles and central moments.

Two compartment modeling

A model, describing pharmacokinetic parameters, is desired (Chapter 2.3) and the two compartment model is fitted to the PET time sequence in each voxel. It is assumed that the perfusion flow is the limiting factor dividing the plasma compartment from the first compartment and that phosphorylation (by hexokinase II) of FDG represents the second limiting factor dividing the first and second compartments. Since dephosphorylation is slow and negligible in the 45 min time span only the phosphorylation is relevant and k4 is assumed to be zero. The blood time activity curve (TAC) is used as the input function, described earlier in this chapter. After experimenting with the curve fitting it is concluded that the calculation of the k values have a tendency to amplify noise in regions with low signal and the input data is smoothed by a boxcar filter to counteract this.

In the fitting, the number of data point surpasses the degrees of freedom and the problem is over determined. The differential equations of the two compartment model are solved numerically and K1, k2 and k3 are calculated in an iterative process. First some initial k values are assumed based on previous experience. According to equation 2.11 the initial values of the k's create a function that convolved with the plasma input function gives the time dependent tissue concentration. The squared difference of the true values and the initial fitted concentration values for each point in time is summed and represents the cost function of the optimization problem. The k parameters are then adjusted following the derivatives of the cost function times a step size defined by the user and the calculations are repeated until changes become exceedingly small. It is important that the initial values are chosen to be close to the true values or a local minimum might be reached instead of the global one. The user defined values are entered through the properties context menu (Figure 14).

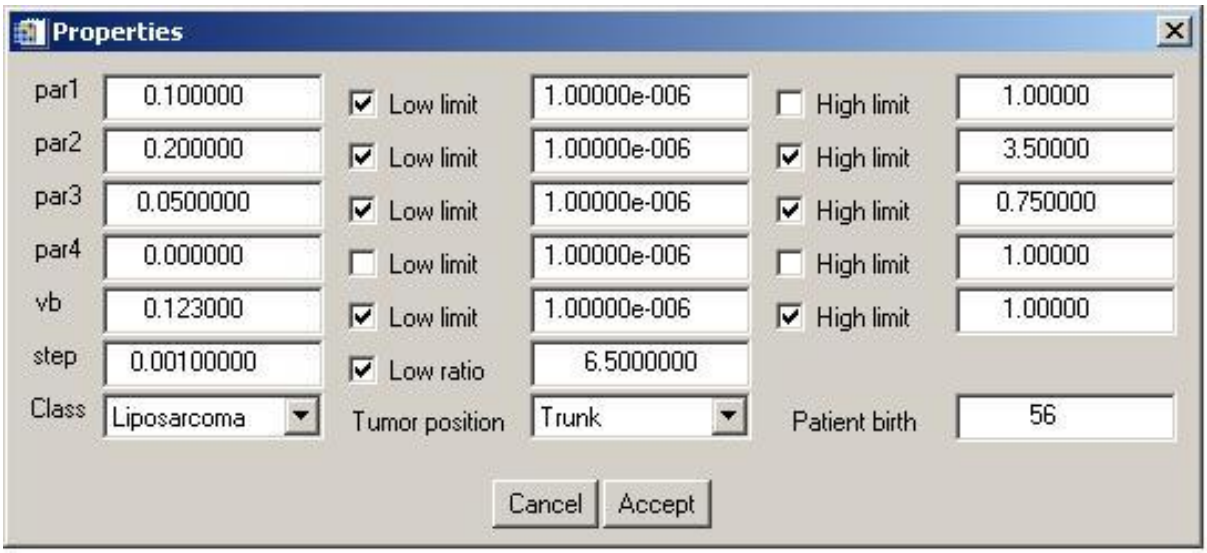

Figure 14. The properties menu. Par1-3 are the K1-3, par4 is an optional k4. Step is the iteration step for the least squares optimization. The check boxes are for turning limits on an off. The low ratio is for adjusting the quality parameter of the plasma sample selector. Class, tumor position and Patient birth are the variables used in classification.

The values in a voxel in the PET time sequence are assumed to be an amalgam of three components: First, FDG present in blood, this is represented as a fraction (vb) multiplied with the plasma function. Second, unphosphorylated FDG (free) in the interstitial or intracellular space, this is FDG not trapped and still free to return to the blood stream. This compartment is calculated from the combined effect of K1, k2 and k3.Third, phosphorylated FDG, bound in the intracellular space. This compartment is the accumulated k3 transfer from the free compartment. Adding the three compartments produces the curve fit (Figure 15). Initially the vascular component contributes the most to the voxel value and as it quickly declines the deposited FDG in the tissue become the dominating factor. In time also this declines as the FDG is either transported out through the bloodstream or trapped, while the trapped component increases until all the other compartments are emptied. As described in Chapter 2.3.2 (Equation 2.15) the metabolic rate (MR_{FDG}) is expressed in k's as $\frac{K_1 k_3}{k_2 + k_3}$ and is calculated for all tumor voxels. As a measure of the quality of fit Pearsons correlation coefficient squared, r^2, is computed between the curve fitting and the original data points.

The result is that for every voxel eight values are calculated: K1, k2, k3, vb, MR_{FDG}, r^2, Patlak slope and Patlak intercept. The data can be displayed as images or composites with CT images and the histograms are described by percentiles and central moments as described in chapter 3.4.

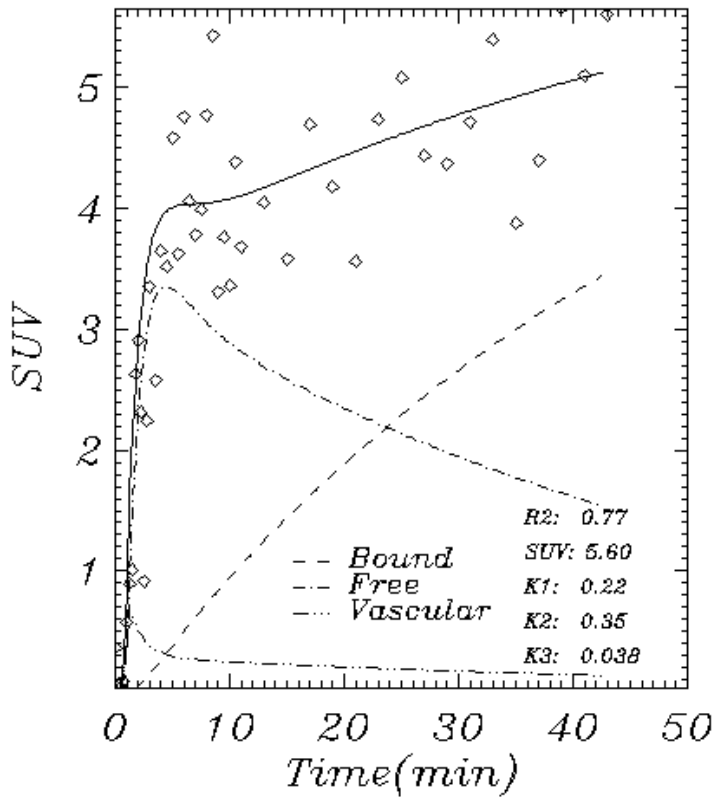

Figure 15. The sample points of a voxel plotted as diamonds. The sum of the three components of the 2 compartment model is shown as a solid. The vascular fraction is drawn as a double dot line. The free, first, compartment is drawn as the single dot line. The bound, second, compartment is drawn as slashed line.

3.6 The patients

Anatomical terms and locations used in this section are found in Appendix A. Of the 11 patients in the study four have a tumor situated in the legs, four in the thorax and 3 three in the hip. The ones in the thorax have a good plasma sampling source in the arch of aorta / thoracic aorta. The aorta is at its widest near the heart where it is about 2-3 cm and multiple voxels (0.5 cm) can be fitted inside. To some extent the ascending arteries (Brachiocelphalic trunk, left common carotid artery, left subclavian artery) can also be used in cranially extended scans, like for patient 5. The hip is second best with parts of the thoracic aorta, abdominal aorta, still visible. Lower, the iliac and later the femoral artery starts showing partial volume effects. The ones situated in the legs have the most partial volume effects as the arteries here are too thin, especially in the tibial arteries. In patient 1 the artery also passes through the tumor and the overlapping part was excluded from the tumor definition. The full list of tumor positions, arteries used for sampling and reference tissue is found in Table 3 while tumor delineations for all patients are shown in Figure 16.

Patient	Tumor Position	Scan area	Plasma sample (number of points)	Tissue equivalent	Tumor delineation	Diagnosis
1	Right leg (Adductor)	Upper legs	Left femoral artery(30)	Left biceps femoris	CT + PET	Liposarcoma/ Leiomyosarcoma
2	Near liver (Medial, close to the lumbar spine)	Abdomen	Thoracic aorta(200)	Erector spinae	Masterplan	Myeloid sarcoma
3	Left buttock (Anterior to gluteus maximus)	Hip / lower abdomen	Abdominal aorta (40)	Right gluteus maximus	CT	Liposarcoma/ Leiomyosarcoma
4	Near left knee joint (Proximal and medial)	Upper legs	Both femoral arteries (50)	Semimembranosus and semitendonous	CT	Liposarcoma/ Leiomyosarcoma
5	Left shoulder blade (Anterior to scapula)	Upper torso	Arch of aorta (65)	Trapezius	Masterplan	Liposarcoma/ Leiomyosarcoma
6	Left calf (gastrocnemius)	Lower legs	Posterior tibial and fibular arteries (30)	Right gastrocnemius	CT	Liposarcoma/ Leiomyosarcoma
7	Right hip (distal anterior to the pelvic girdle)	Hip / upper legs	Femoral artery (75)	Left rectus femoris	CT + PET	Liposarcoma/ Leiomyosarcoma
8	Right hip (ventral to pubic bone)	Hip / upper legs	External iliac (15)	Right rectus femoris	CT	Liposarcoma/ Leiomyosarcoma
9	Right knee (proximal end of fibula)	Lower legs	Posterior tibial (15)	Gastrocnemius	Masterplan	Schwannoma
10	Shoulder (proximal end of humerus)	Upper torso	Arch of aorta and descending aorta (250)	Triceps brachii and posterior part of the deltoid	Masterplan	Liposarcoma/ Leiomyosarcoma
11	Back (medial of scapula)	Torso	Thoracic aorta (400)	Trapezius	CT + PET	Liposarcoma/ Leiomyosarcoma

Table 3. List patient data and diagnosis. Plasma sample and tissue equivalent are the respective locations. Scan area is the region of the dynamic PET scan. Tumor delineation is the method used for tracing the tumor borders.

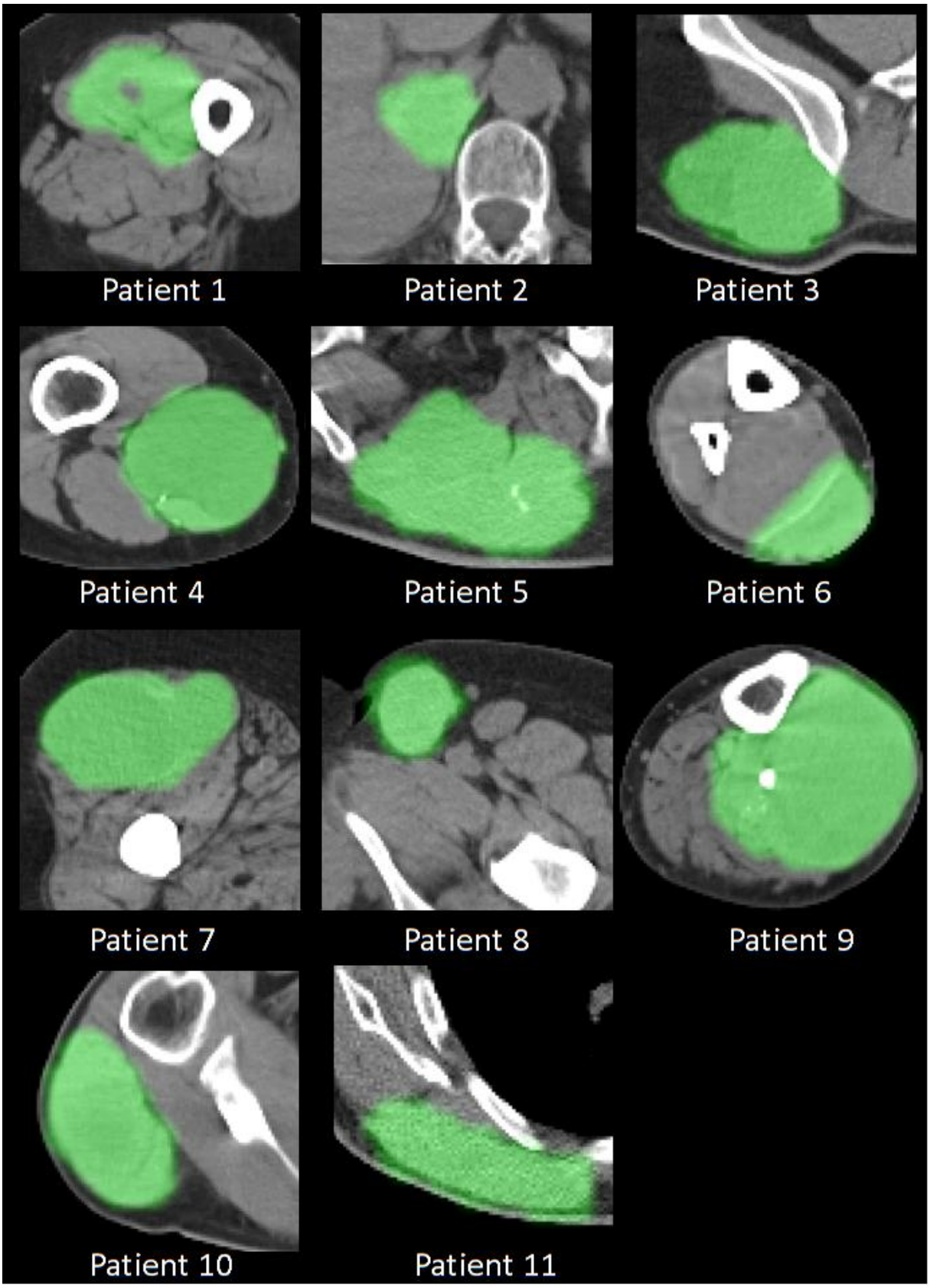

Figure 16 Tumor delineations of patients 1-11. The tumor regions are shown as green transparent areas.

The Masterplan data is used for delineation of all tumors when it is available. Otherwise the CT is used when the tumor is visible against the surrounding tissue, usually fat. But in some tumors like patient 1 there is no clear boundary. The texture of the tumor is slightly different from the surrounding muscle, a bit smoother, but we need to supplement with the PET scan to see the general shape. For patient 2, the tumor is situated on/near a tumor bed where a tumor has been resected earlier. This complicates the identification as the scar tissue also absorbs FDG and the liver has a structure similar to the tumor.

Patients 2, 5, 9 and 10 where subjected to radiation therapy. Patient 2 is unique in having two sequences prior to therapy. Patient 5 and 9 have one dynamic sequence missing. Patient 5 and 10 had an issue with tumor delineation as in the Masterplan CT images the patients keep their arms high, representing the setting during the radiation therapy. In the PET scan, however, arms are kept low, since this is the normal procedure. This slightly alters the position of the tumor leading to some subjectivity when the tumor delineation is transferred from one system to the other.

3.7 Mann-Whitney

To check for statistical significance of data refuting a null hypothesis like: "The distribution of class 1 and class 2 are equal" a statistical hypothesis test is required. As the distributions of the data used in this thesis are unknown tests assuming a certain distribution, like the Student's t test, may not be appropriate. Mann-Whitney is a test similar to the Student's t test though it does not assume any particular distribution. It was proposed initially by Frank Wilcoxon in 1945 but was further developed by Henry Mann and Donald Ransom Whitney. Since the exact parameter composition of our test is still unknown several parameter compositions are attempted. Statistical significance tests are designed to keep the chance that a certain result is a chance finding below a certain threshold. Testing for double the number of parameters doubles the probability of a chance finding and a common adjustment is reducing the significance level by the number of parameter combinations (Bonferroni). The number of patients in the data set is only 11 and the best distribution possible is with 6 patients in one group and 5 in the other. This distribution has the highest possible p value of 0.01 with all points of each distribution separated from the other group. This means that for a perfect match 5 parameters could be tested and still be termed statistically significant ($\alpha < 0.05$). However with an uneven group, like 9-2, the best possible p value is 0.05. For this distribution it is

impossible to conclude anything with a statistical significance for anything more than a single parameter setup.

In the current work only parameter selection will be considered. The validations of the parameters have to be conducted in a separate study and the results are reported in the rank based U value of the Mann-Whitney test rather than including the significance level to avoid any confusion. However the level that would indicate a value of statistical significance is noted and used as an indication of a possible correlation (Appendix B). The U value is simply a rank based value, summing the number of samples in one class that is lower than the samples in the other class. In other words the highest value in the case of 9 patients in one class and 2 in the other is 0 or 18. And as the null hypothesis is that they are equal such a result indicates that they are not. From a significance point of view there are no difference in a 0 or 18 value and values below 9 is mirrored (18 – value) giving a range of 9-18. For interpreting the results it does matter however and both values are used.

The user enters class information like diagnosis, patient age and tumor position into the program through the properties menu (Figure 14). From these opposing class lists matrixes of U values are created with twelve parameters and their percentile and central moment descriptors. The twelve parameters used are the SUV at 2 and 45 minutes (SUV^E and SUV^L), unscaled or scaled relative to the uptake in normal tissue (SUV_r^E and SUV_r^L), the three k values K1,k2 and k3, the vascular fraction (vb) , the metabolic rate (MR_{FDG}), Patlak slope and intercept and the curve fitting quality (r^2). The r^2 might not in itself have any biological meaning but it may be linked to e.g. necrotic areas with low activity and low signal to noise ratio.

Part II
Results and discussion

4 Results and analysis

4.1 Visual inspection

During the first minute after injection the FDG is still concentrated to a part of the blood volume and can be seen propagating from the injection point and spreading through the body. Its arrival to the tumor can reveal the blood vessels supplying the tumor, as in this phase the blood vessels themselves contribute the most to the image intensity. Examining the three first images of Patient 10 (Figure 17) reveals that the injection is initially seen in the subclavian vein. Then the initial bolus of FDG is transported to the heart and out into the lungs for oxygenation. This is seen in the second frame (30s) where the lungs have their strongest signal. The oxygenated blood return to the heart and is then distributed in the body. A typical resting male has a stroke volume of 70ml/beat and about 75heartbeats/min which mean that after about a minute the full 5liters of blood have been circulated. Thus, already in the image 45 seconds after injection the FDG has reached the entire upper torso and the first signal inside the tumor is detected. The slices are of the entire body but to obtain good contrast in the tumor, which is the region of interest, the value range is limited by the user to that of the tumor. This means that other structures of higher values loose their texture and get limited to the maximum value of the tumor (the image appears to be “burnt through”).

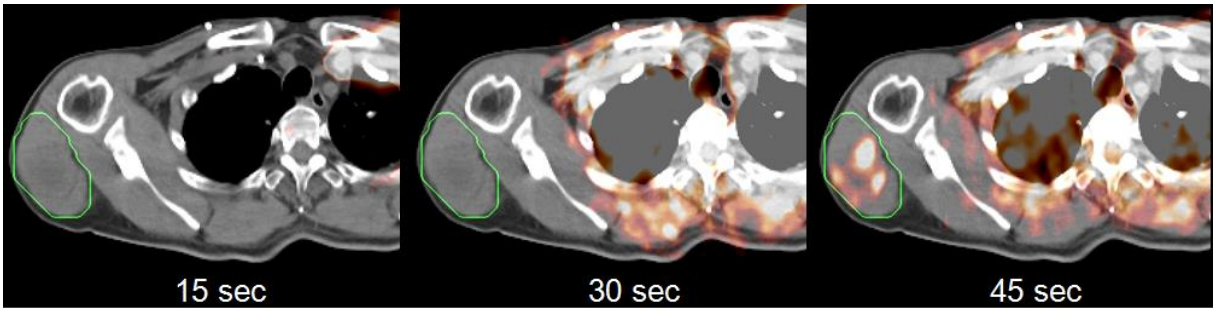

Figure 17. The bolus phase of patient 10. From left to right, the panels show SUV images sampled 15, 30 and 45s, respectively, after injection. All images are blended with the CT frame. The tumor delineation is indicated with a green contour.

In patient 9 the three first images (Figure 18) reveal a much more gradual development. This is a tumor in the extremity and no other high intensity structures are seen and circulation is weaker. In the image 15 seconds after injection the tibial aorta contributes the most to the image intensity and in the image 30 seconds after injection the FDG is diluted into a vascular

region in the left part of the tumor. Between the images 30 and 45 seconds after injection, little difference can be seen.

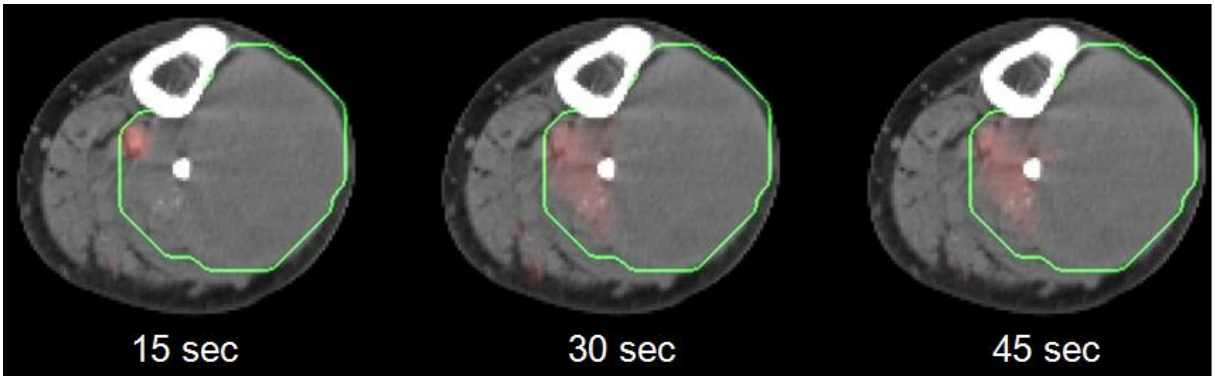

Figure 18. The bolus phase of patient 9. From left to right, the panels show SUV images sampled 15, 30 and 45s, respectively, after injection. All images are blended with the CT frame. The tumor delineation is indicated with a green contour.

As the FDG is distributed evenly in the blood the high initial concentration evens out at more stable lower value. The next phase can be seen as a vascular phase where FDG in the blood is transported into the tissue and distributed. In this phase the deposition of FGD to the tissue depend on the diffusion in the tissue and perfusion of the vasculature. The concentration of FDG in the blood falls as it is distributed into the tissue, and the blood vessels contribute less to the image intensity. The further this effect progresses the less distinct the image becomes.

Images from patient 10 1-4 minutes after injection (Figure 19) reveal that the lungs and blood vessels are less dominant here than in the images during the first minute while in the tumor there is still a good correlation with the 45 sec image. The strong initial signal in the tumor is consistent with the leaky nature of vasculature generated through angiogenesis though the distribution is heterogeneous and also areas of poor circulation are apparent. When the FDG concentration in blood falls as the FDG is absorbed into the tissue this effect loses its prominence and the tumor appears weaker and more diffuse.

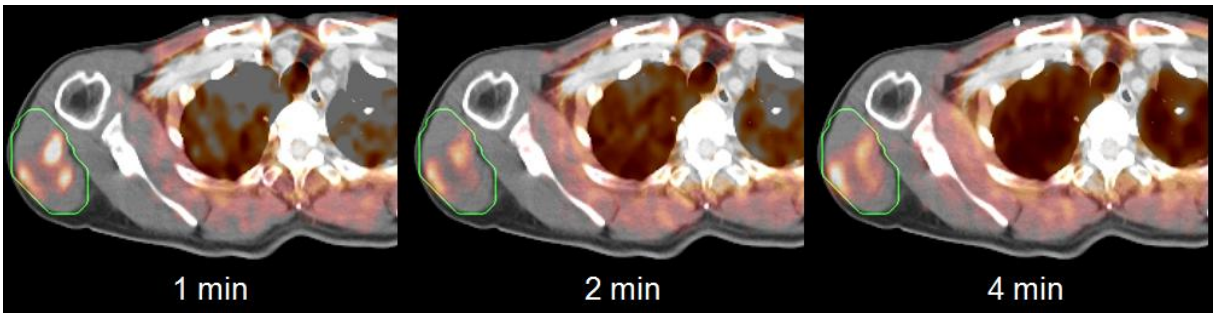

Figure 19. The vascular phase of patient 10. From left to right, the panels show SUV images sampled 1, 2 and 4min, respectively, after injection. All images are blended with the CT frame. The tumor delineation is indicated with a green contour.

Images from patient 9 1-4 minutes after injection (Figure 20) show a gradual increase in image intensity. The differences are subtle but an elliptical dark central area is starting to be visible, possibly an area of hypoxia and necrosis.

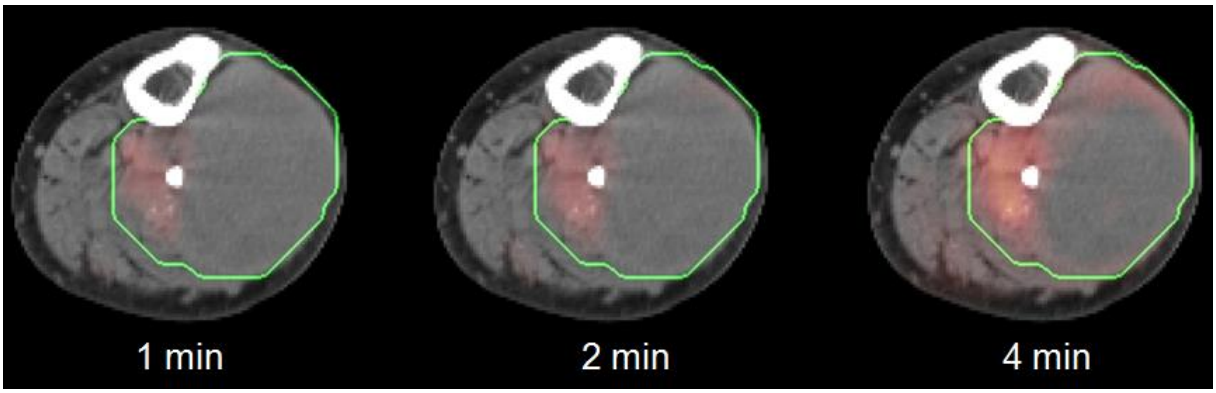

Figure 20. The vascular phase of patient 9. From left to right, the panels show SUV images sampled 1, 2 and 4min, respectively, after injection. All images are blended with the CT frame. The tumor delineation is indicated with a green contour.

Gradually a semi, 'transient', steady state develops where slowly the FDG is filtered from the blood by the kidneys. In the kidneys the FDG is recognized as waste and is secreted through the urinary canals into the bladder. In this way, FDG is gradually cleared from the blood. The FDG already deposited in the tissue either leaks back into the blood or gets trapped metabolically inside the cells. Slowly the contrast of metabolic areas to the surrounding tissue is established. At this point the bladder and kidneys have a strong signal along with tumor cells, the spine and involuntary muscles like the heart.

It can be seen in the last images of patient 10 (Figure 21) that the signal from the blood in the lungs and arteries gradually decrease. Trapped FDG still produces a signal in the tumor and

metabolically active areas like the nervous system (spine), though changes at this point are slow and hard to detect visually.

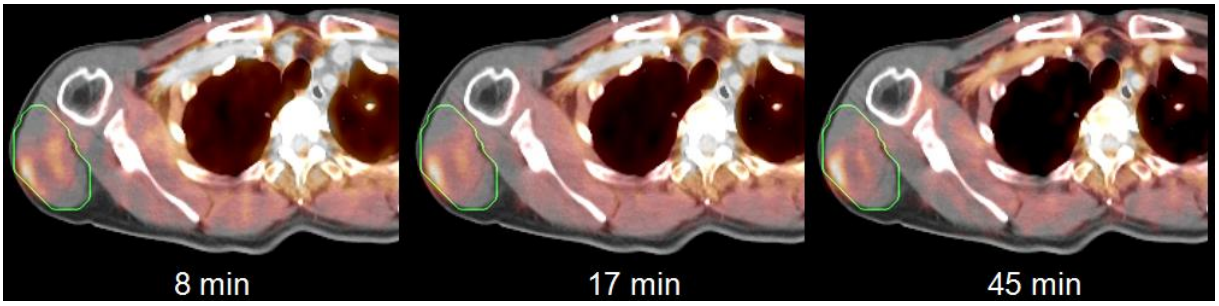

Figure 21. The metabolic phase of patient 10. From left to right, the panels show SUV images sampled 8, 17 and 45min, respectively, after injection. All images are blended with the CT frame. The tumor delineation is indicated with a green contour.

In the last images of patient 9 the earlier trend continues and the tumor intensity gets more pronounced. And while most of the tumor displays some activity the leftmost region, where the initial vascular phase was the most active, is still the dominant region. Accumulation in normal tissue is hardly present.

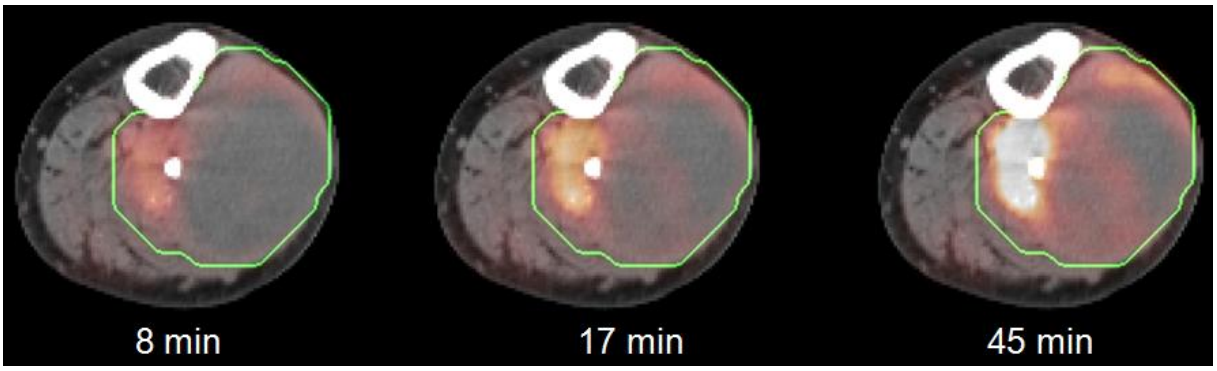

Figure 22. The metabolic phase of patient 9. From left to right, the panels show SUV images sampled 8, 17 and 45min, respectively, after injection. All images are blended with the CT frame. The tumor delineation is indicated with a green contour.

4.2 Histogram analysis

The images are reduced, as explained in chapter 3.4, to histograms for further analysis. Examining the histograms of the images of patient 10 (Figure 23) reveals that one minute after injection, in the bolus phase, a maximum SUV of about 3 is obtained. This is the effect of the high concentration arterioles producing small peaks of activity. The histogram is shifted heavily towards the lower values as large parts of the tumor are unsupplied and have values

close to zero. In the histogram 2 minutes after injection, the vascular phase, the maximum SUV is roughly 2.5. The histogram has more or less the same shape as after one minute but it has a much more developed intermediate region as the tissue starts to absorb the FDG. In the histogram 8 minutes after injection the high values are less pronounced as the blood is diluted and the lowest values have disappeared. The histogram 45 minutes after injection, corresponding to the metabolic phase, shifts the shape towards the original histogram. It has fewer values in the middle as the FDG is cleared out by the blood while the top values, representing trapped FDG, are slightly strengthened.

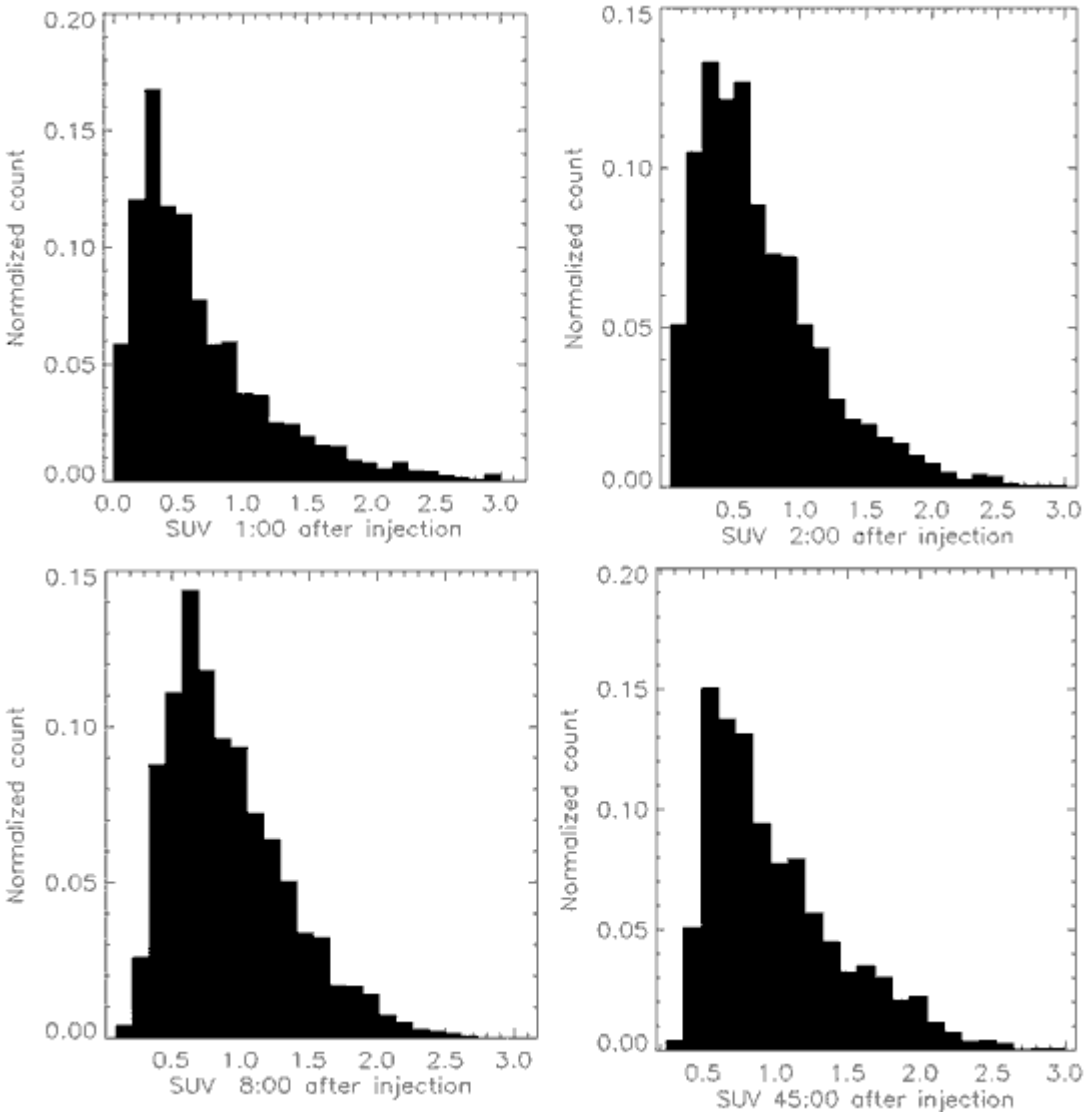

Figure 23. SUV histograms of the tumor in patient 10. From top left to bottom right, the panels show histograms generated 1, 2, 8, and 45 minutes, respectively, after injection.

There is one histogram for every time point in the imaging protocol (chapter 3.2) and the values extracted from the histograms can be plotted in time activity curves (TAC). Examining

the TAC's from patient 10 (Figure 24) reveals that the mean value is more or less constant after 10 minutes, which could be seen visually in Figure 21.

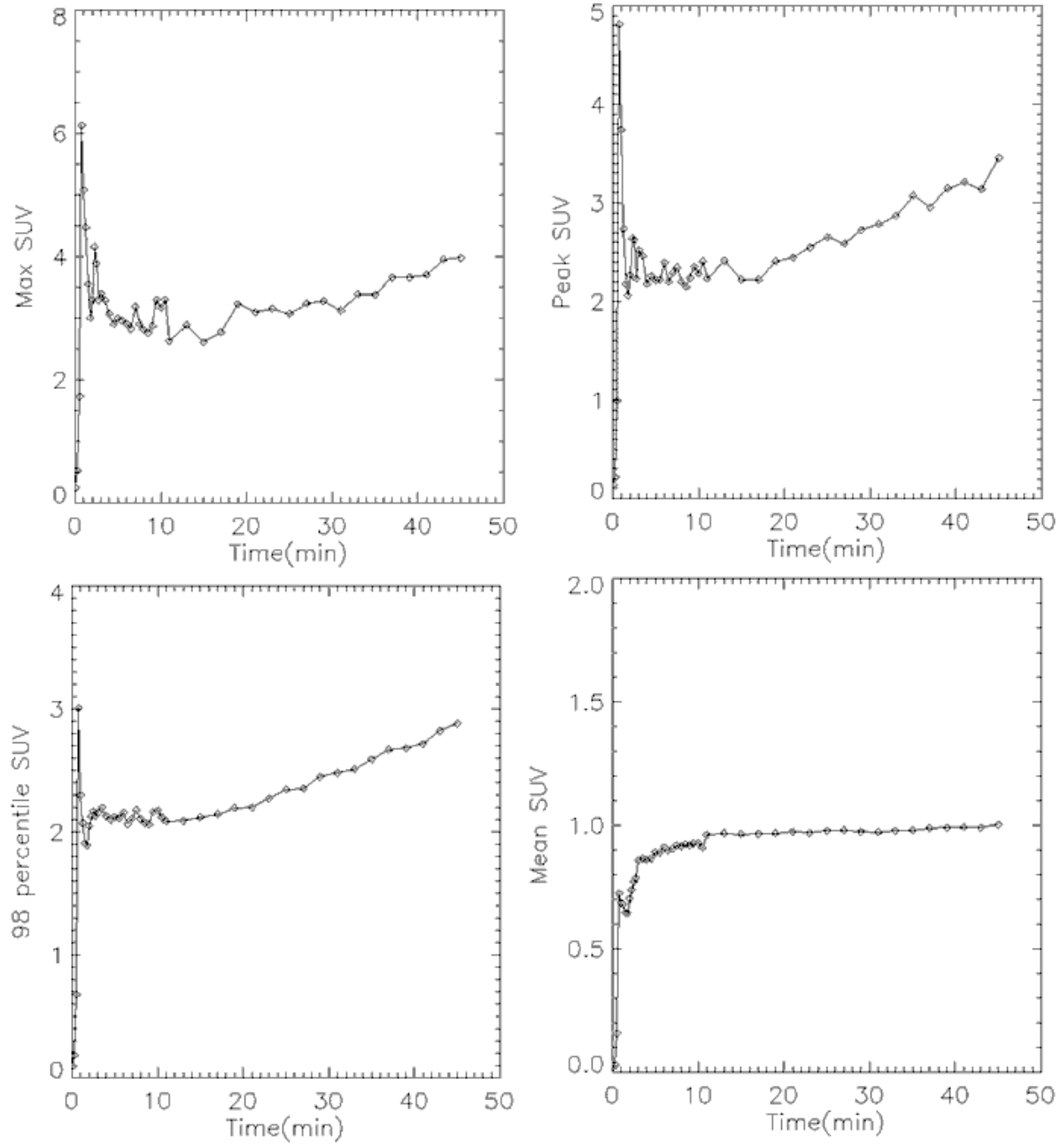

Figure 24. Time activity curves (TAC's) for the tumor in patient 10. From top left to bottom right, the panels show TAC's for the maximum, Peak, 98th percentile and mean SUV, respectively.

The 98th percentile TAC increases with time so in effect the most active part of the tumor is still accumulating FDG, but as the tumor delineation has included normal or non aggressive tissue as well this part is in decline and in total they even out producing a flat mean tumor value. The Max value and Peak value also show an increase, but as they sample fewer points they have a higher variation in their signal. The Max value is naturally higher than the Peak

value but the slope is about the same. The 98th percentile on the other hand has a lower slope as less active voxels have been included. Both the mean and the 98th percentile have smoother TAC's than the Peak and Max.

4.3 Parametric analysis

The k values are also produced as images, though as 3D sequences rather than 4D sequences, as the time dimension was used to produce the k values. The parametric images, in a similar way as the SUV values, are inspected visually. Examining the k images from patient 10 (Figure 25) reveals that value range in the whole body is larger than the value range in the tumor. The lungs show up as regions with high K1 and k2 while k3 is low. This is consistent with the lungs having a high plasma exchange rate but little glycolysis. The spine displays high K1, k2 and k3 values which is consistent with its high dependence on glycolysis and blood sugar. Artifacts near the resting points of the patient can be seen, possibly due to reduced blood circulation in the region leading to a curve fitting based on noise.

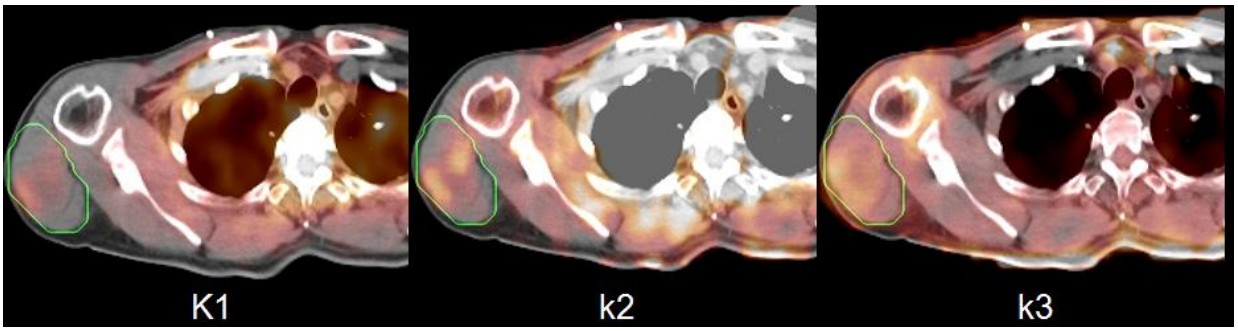

Figure 25. Image of the k parameters of patient 10 alpha blended with the corresponding CT image. From left to right the panels show K1, k2 and k3 images. The tumor delineation is indicated with a green contour.

As seen in the k images of patient 10 (Figure 25) there are some artifacts in the k images that require some examination. The computer program allows each voxel to be inspected manually (Chapter 3.4) by plotting the compartmental curves and the total fitted curve. The two compartment fitting can adapt to most normal tissue and tumor voxels (Figure 26), whether they have a pronounced early uptake or not.

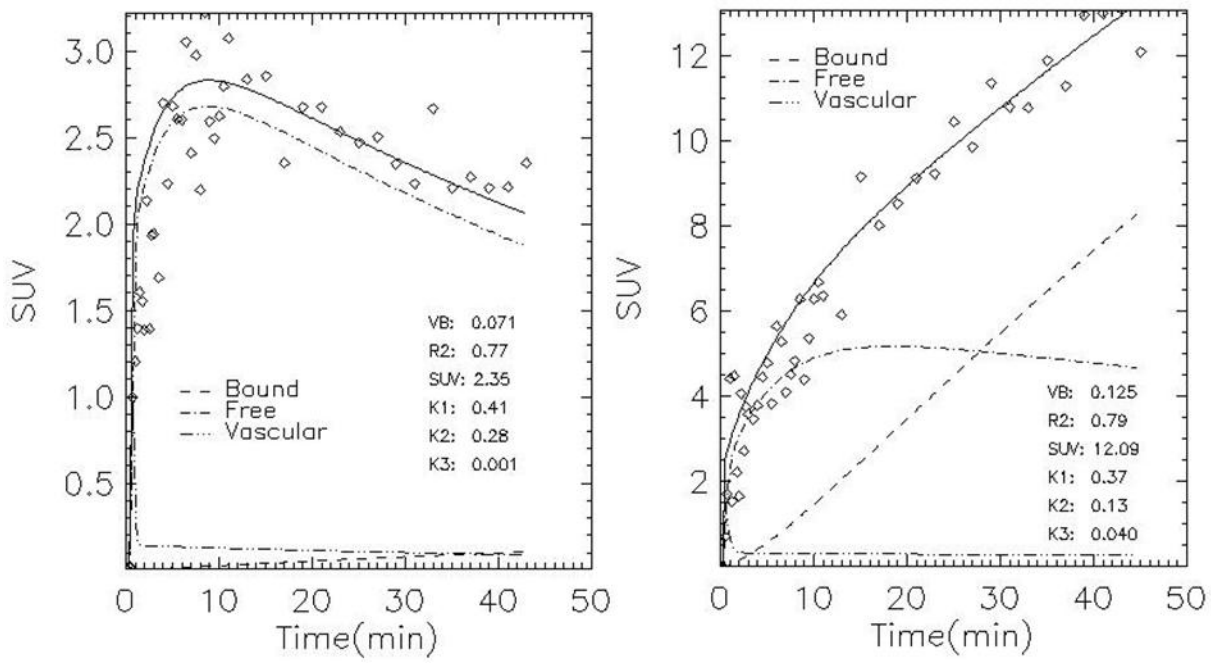

Figure 26. Curve fitting of tumor voxels. To the left a curve fitting of a voxel from patient 4 with an initial peak followed by a declining curve is plotted. To the right a curve fitting of a voxel from patient 8 with a strictly increasing curve is plotted.

Examining some of the points at the edge of patient 10's tumor reveal a few voxels with a low, noisy signal with neither an initial peak nor a distinct washout but rather a weakly increasing signal (Figure 27). The fitting is poor, the mismatch between K1(0.03) and k2 (1.22) is improbable and the k3 value is artificially large compared to the also steadily increasing plot in the previous example (Figure 26). The low K1 and high k2 mathematically ensures a low free compartment which in turn allows the high k3. Points like this are uncommon in the tumor itself, but can be found on the tumor edge or in full body images. Other artifacts are seen when the voxel concentration curves are dominated by an artery. From the image of patient 10 (Figure 25) a region in the top left can be identified as the source of the bolus. Selecting a voxel from this region gives a curve fitting closely resembling the plasma function (Figure 27). Since voxel development resembles the input function the curve fitting is simply finding the correct vascular fraction. The vascular fraction is then multiplied by the plasma function to produce a curve fit matching the tissue values. If the curve fit is already at its optimum at that point K1 and k2 can be put to approximately zero. With the free compartment close to zero k3 is free to assume a rather arbitrary value (in this case 1.8 compared to a normal k3 value of 0.001).

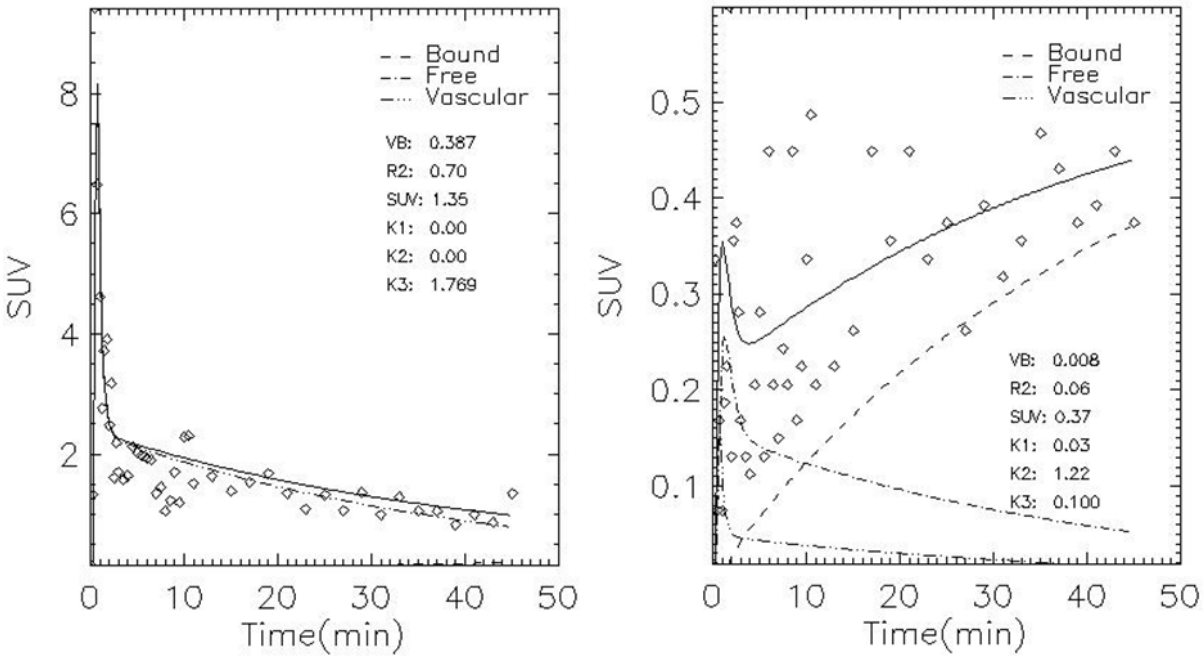

Figure 27. Example of the two most common artifacts. To the left is a plot of a voxel near the injection artery in patient 10. To the right is a tumor border voxel from patient 10.

As stated in chapter 2.3.1; the pharmacokinetic model is composed of a free compartment, a bound compartment and a plasma compartment. The plasma compartment is the plasma function multiplied with vb, the vascular fraction, and an image of patient 10's vb is seen to correlate with the arteries (Figure 28).

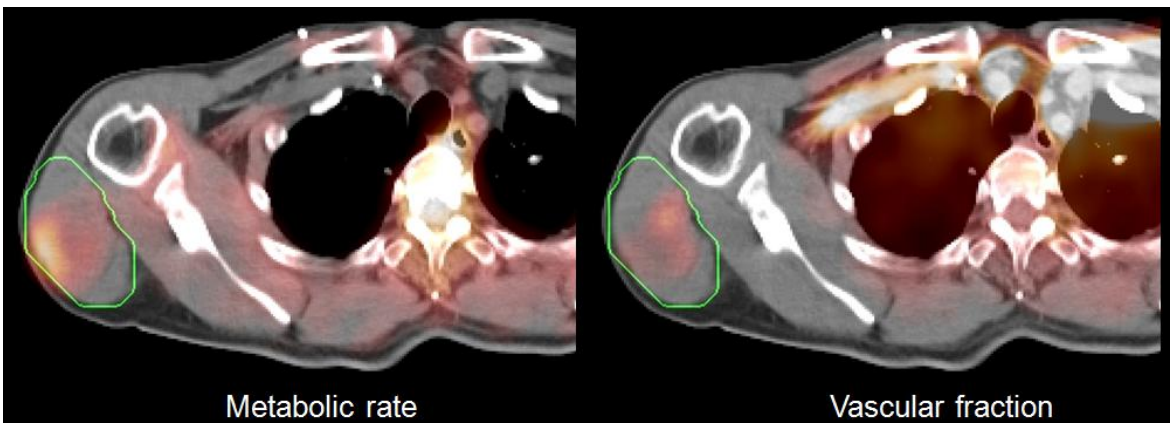

Figure 28. The metabolic ratio and vascular fraction of patient 10. The images show the metabolic rate (left) and vascular fraction (right), blended with the CT image. The tumor delineation is indicated with a green contour.

In Figure 28, the oblong shape to the left is the (right) subclavian artery. The large dot in the middle is the brachiocephalic trunk. The two smaller dots are the left common carotid and left subclavian. The lungs have a large vascular fraction and compared to normal tissue the tumor has an increased vascular fraction. The image of patient 10's metabolic rate (Figure 28) shows

a correlation with the spine and tumor while the musculature displays a weaker signal. It can also be seen a somewhat negative correlation between the vascular fraction and the metabolic rate.

Just as with the SUV values the k values are reduced to histograms. From patient 10's histograms (Figure 29) it can be seen that the shapes of the k histograms are similar to each other, not really Gaussian but rather leaning towards the lower values. The k2 is the widest, has the highest mean value and the most distinct outliers. K1 is about a fifth of k2 while k3 is less then a tenth of k2. The low k3 shows that there is a small fraction of the available FDG that is actually absorbed at a time.

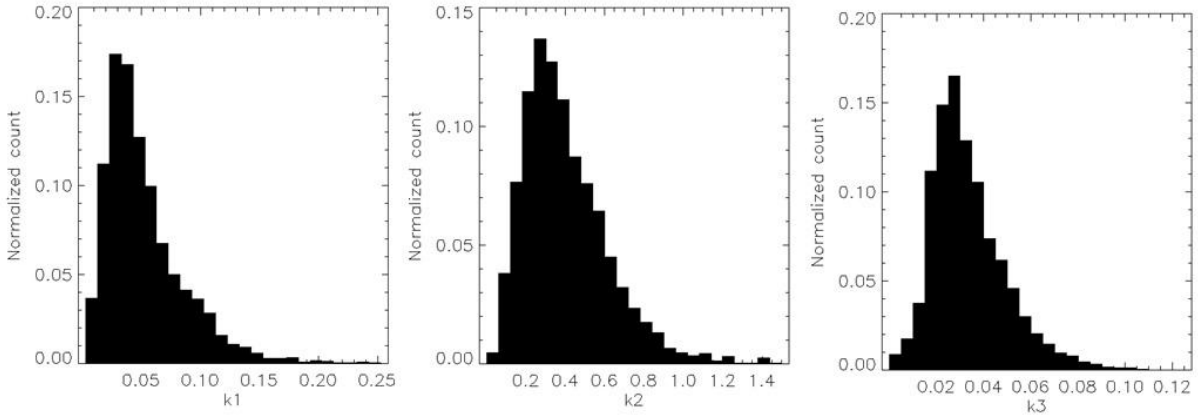

Figure 29. The k histograms of the tumor region of patient 10. From left to right the histograms of K1, k2 and k3, respectively, are shown.

The histogram of the vascular fraction of patient 10 (Figure 30) is quite similar to the k histograms. The histogram of patient 10's metabolic rate (Figure 30) on the other hand has more the shape of a right angled triangle.

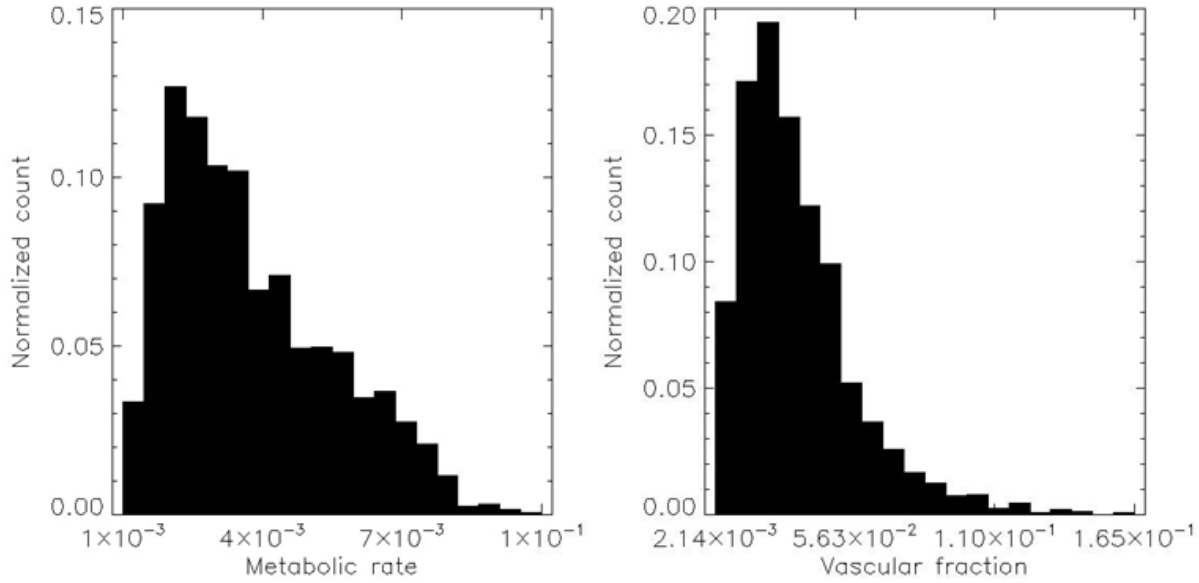

Figure 30. To the left a histogram of the metabolic rate parameter calculated from the k values in the tumor of patient 10 is displayed. To the right the histogram of the vascular fraction of the same patient is displayed.

A Patlak analysis (Chapter 2.3.3) is computationally much faster than the two compartment model and produce similar images. Examining the Patlak images from patient 10 (Figure 31) reveal that the Patlak slope resembles the image of the metabolic rate and the Patlak intercept resembles the image of the vascular fraction (Figure 28). However the images have less contrast and more noise compared to the normal tissue.

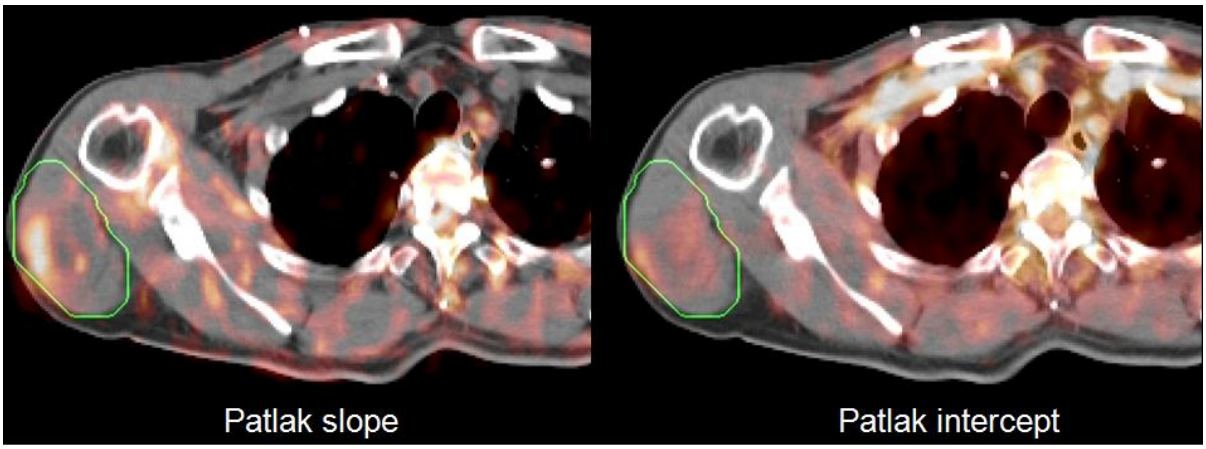

Figure 31. Patlak images of patient 10. To the left the Patlak slope is composited with the CT image and to the right the Patlak intercept is composited with the CT image. The tumor delineation is indicated with a green contour.

The Patlak slope histogram of patient 10 display a shape similar to the k parameters while the histogram of the intercept is almost Gaussian (Figure 32)

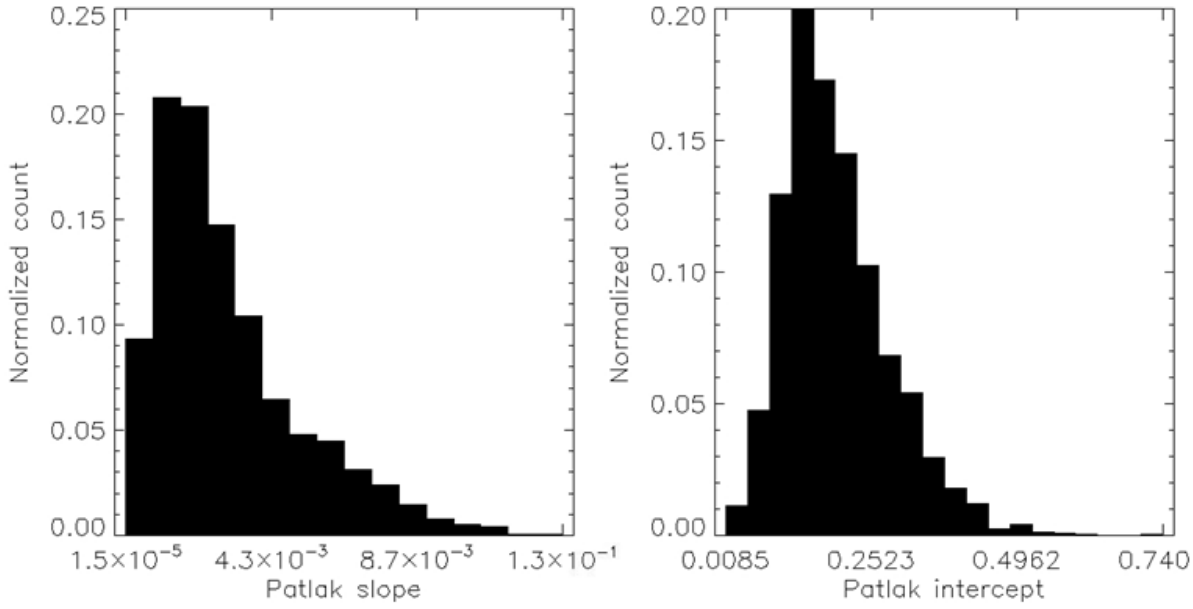

Figure 32. The Patlak histograms of the tumor in patient 10, showing the Patlak slope (left) and intercept (right).

4.4 Metabolic rate

It is assumed that the cellular glycolytic activity is the most relevant attribute to PET cancer diagnostics and as such a comparison between the three different measures for metabolic activity is timely. First, we have the direct, static SUV image obtained 45 minutes post injection. Second, we have the graphical solution for the endpoint, the Patlak slope. Third, we have the two compartment solution with the metabolic rate (MR_{FDG}). Comparing these images for patient 1 (Figure 33), patient 5 (Figure 37), patient 6 (Figure 38), patient 8 (Figure 40) and patient 9 (Figure 41) reveal that the parameters display a similar intensity distribution and all have good contrast to the surrounding tissue. The images of patient 2 (Figure 34) show that the liver masks the tumor in both the image of the SUV at 45 min (SUV^L) and in the Patlak slope image by having the same level of activity as the tumor. In the MR_{FDG} image the tumor is indeed visible as the liver is found to be metabolically inactive. The images of patient 3 (Figure 35), patient 4 (Figure 36) and patient 7 (Figure 39) show good contrast for the SUV^L and metabolic rate while the Patlak slope has a weaker contrast to the normal tissue. In patient 7 the SUV^L delineates the entire tumor in a better way than the MR_{FDG} which displays a more heterogeneous distribution. The images of patient 10 (Figure 42) and patient 11 (Figure 43) indicate that SUV^L and MR_{FDG} have better contrast than the Patlak slope while that Patlak slope and MR_{FDG} to some degree eliminate arteries (subclavian and heart) from the image.

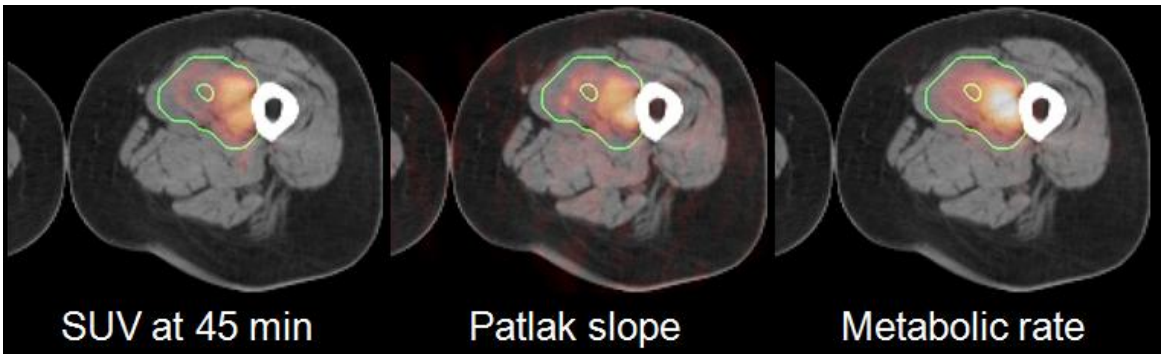

Figure 33. Comparison of SUV(left), Patlak slope(middle) and metabolic rate(right) in patient 1. All images are blended together with the CT image. The tumor delineation is indicated with a green contour.

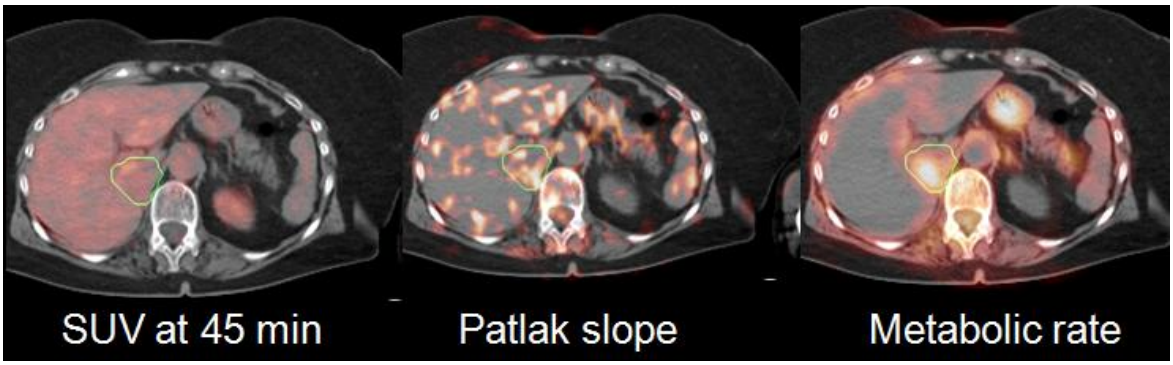

Figure 34. Comparison of SUV(left), Patlak slope(middle) and metabolic rate(right) in patient 2. All images are blended together with the CT image. The tumor delineation is indicated with a green contour.

Figure 35. Comparison of SUV(left), Patlak slope(middle) and metabolic rate(right) in patient 3. All images are blended together with the CT image. The tumor delineation is indicated with a green contour.

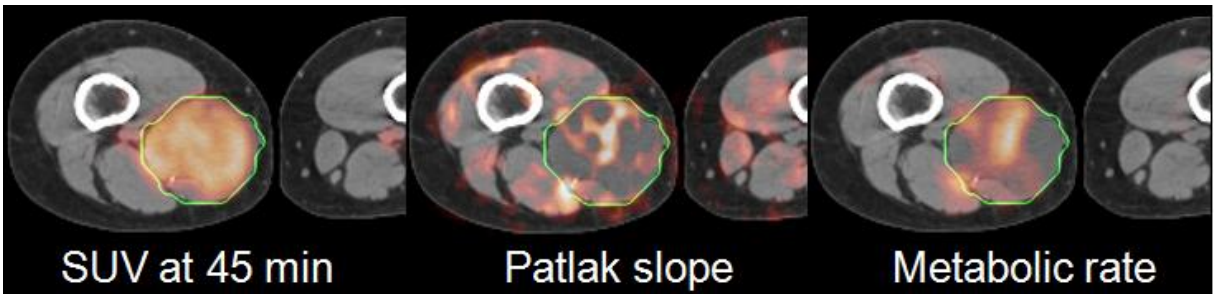

Figure 36. Comparison of SUV(left), Patlak slope(middle) and metabolic rate(right) in patient 4. All images are blended together with the CT image. The tumor delineation is indicated with a green contour.

Figure 37. Comparison of SUV(left), Patlak slope(middle) and metabolic rate(right) in patient 5. All images are blended together with the CT image. The tumor delineation is indicated with a green contour.

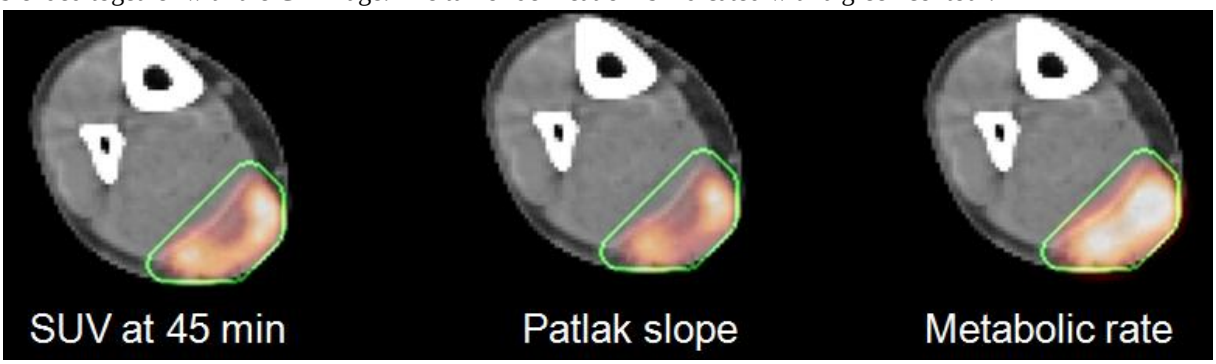

Figure 38. Comparison of SUV(left), Patlak slope(middle) and metabolic rate(right) in patient 6. All images are blended together with the CT image. The tumor delineation is indicated with a green contour.

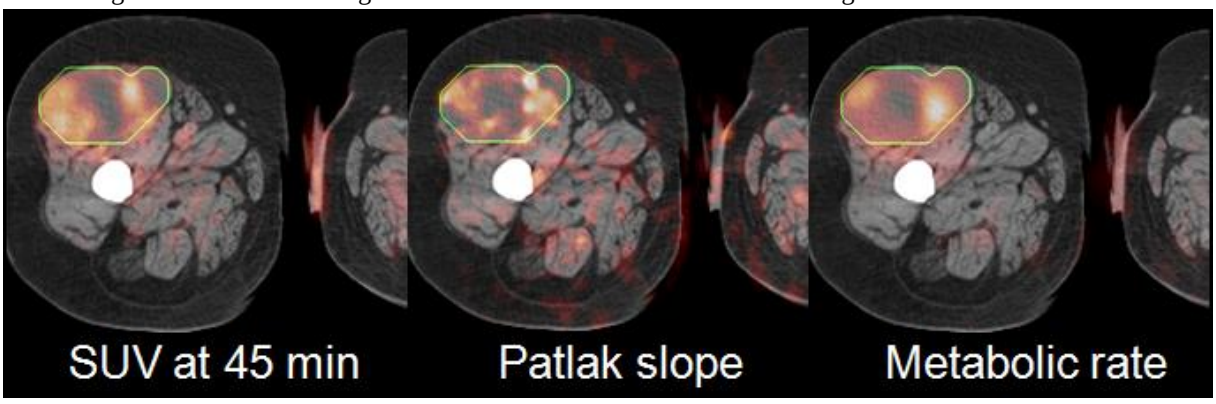

Figure 39. Comparison of SUV(left), Patlak slope(middle) and metabolic rate(right) in patient 7. All images are blended together with the CT image. The tumor delineation is indicated with a green contour.

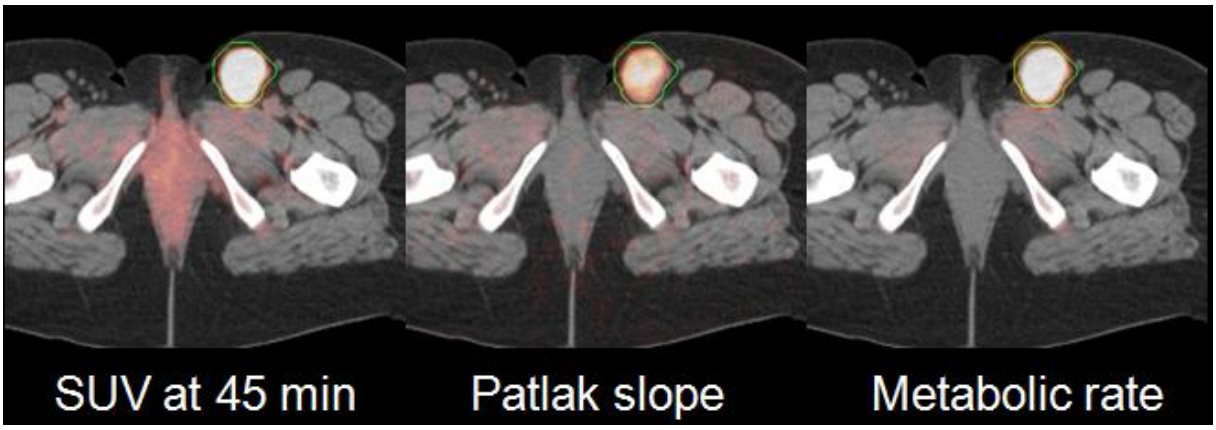

Figure 40. Comparison of SUV(left), Patlak slope(middle) and metabolic rate(right) in patient 8. All images are blended together with the CT image. The tumor delineation is indicated with a green contour.

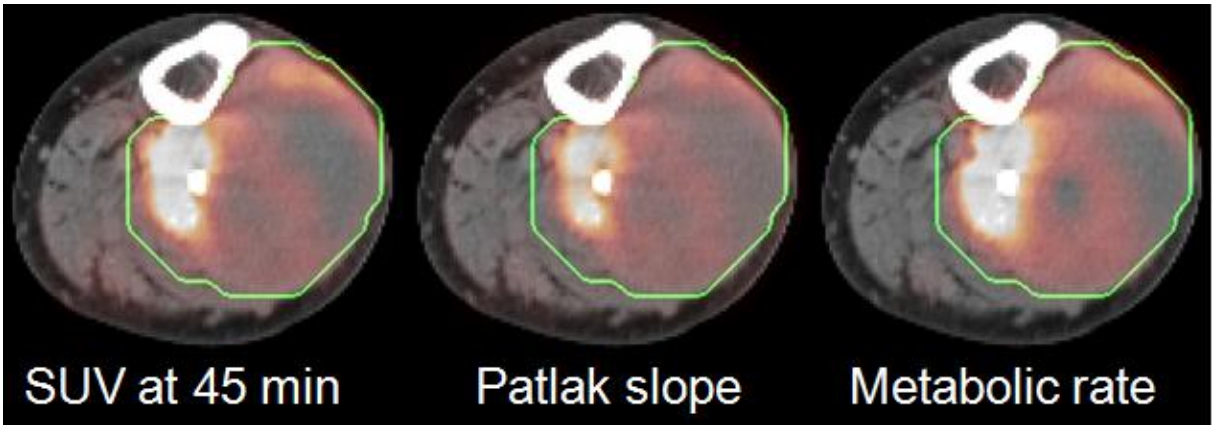

Figure 41. Comparison of SUV(left), Patlak slope(middle) and metabolic rate(right) in patient 9. All images are blended together with the CT image. The tumor delineation is indicated with a green contour.

Figure 42. Comparison of SUV(left), Patlak slope(middle) and metabolic rate(right) in patient 10. All images are blended together with the CT image. The tumor delineation is indicated with a green contour.

Figure 43. Comparison of SUV(left), Patlak slope(middle) and metabolic rate(right) in patient 11. All images are blended together with the CT image. The tumor delineation is indicated with a green contour.

4.5 Classification

The full data of all the patients are found as scatter plots in Appendix C. Here some correlations between the parameters can be seen. Computing the voxel correlation between two variables and taking the mean of all the tumors strengthens this observation (Table 4).

	Pat_i	Pat_s	MR_{PET}	vb	k3	k2	K1	SUV^L
SUV^E	0.42	0.19	0.24	0.40	0.16	0.19	0.64	0.38
SUV^L	0.28	0.56	0.55	0.24	0.18	0.10	0.36	
K1	0.31	0.22	0.33	0.34	0.22	0.44		
k2	0.08	0.08	0.16	0.18	0.15			
k3	0.15	0.31	0.53	0.10				
vb	0.29	0.11	0.16					
MR_{PET}	0.13	0.57						
Pat_s	0.20							

Table 4 The mean voxelwise correlations between the PET parameters. Pat_s is the Patlak slope, Pat_i is the patlak intercept. The color grades indicate values from 0.4 to 0.7.

The early SUV has the strongest correlation with the K1 parameter which describes the wash-in, and although k2 is correlated with K1 it is not correlated with the early SUV. The vb and patlak intercept also show some correlation with the early SUV, though not with each other. The late SUV is correlated with MR_{PET} and patlak slope, which in turn are correlated with each other, while k3 is correlated with MR_{PET} but not with the late SUV.

Operating on all the voxel values is impractical and the rest of the analysis will focus on histogram values (Table 5). All the patients except patient 2 show an increase in SUV from early to late. This is somewhat reversed for the SUV_r where all the patients except 3, 8 and 9 show a decrease in SUV_r form late to early. The patients are divided into two groups according to four different criterions. First, the patients are divided according to tumor diagnosis with liposarcoma and leiomyosarcoma in one group and myeloid sarcoma and schwannoma in the other group. Second, the patients are divided according to whether the patient is born before or after 1960. Third, the patients are divided according to whether tumor size is greater than $200cm^3$. Fourth, the patients are divided according to whether the tumor is situated in the extremity or in the trunk.

	SUV^E	SUV_r^E	SUV^L	SUV_r^L	K1	k2	k3	Met	Pat	vb
Patient 1	3.8	9.4	5.6	7.9	0.8	0.9	0.04	0.034	0.06	0.21
Patient 2	6.0	8.8	3.8	3.7	1.1	1.6	0.02	0.009	0.01	0.21
Patient 3	1.7	3.9	2.5	4.2	0.2	0.6	0.03	0.01	0.02	0.03
Patient 4	2.4	9.4	3.3	4.7	0.4	0.3	0.01	0.009	0.01	0.09
Patient 5	3.2	7.3	7.4	7.1	0.2	0.4	0.07	0.023	0.03	0.11
Patient 6	3.0	32.9	5.3	8.4	1.8	2.2	0.03	0.03	0.05	0.30
Patient 7	4.4	5.3	4.9	3.7	0.9	1.1	0.02	0.012	0.02	0.08
Patient 8	5.2	5.9	18.0	12.2	0.4	0.4	0.06	0.094	0.10	0.14
Patient 9	1.9	4.5	10.5	14.8	0.4	1.4	0.22	0.084	0.12	0.19
Patient 10	2.1	5.1	2.8	3.4	0.1	0.6	0.06	0.008	0.01	0.10
Patient 11	4.5	3.5	4.7	2.8	0.4	2.4	0.10	0.011	0.01	0.08

Table 5. Summary of the 11 patient's most important values. SUV^E means the SUV value from the 1:30-2:45 timeframe. SUV_r^E is the same value normalized to the reference tissue. SUV^L is the SUV value from the 43-45 timeframe and the respective scaled value is the SUV_r^L. All SUV values are the 98^{th} percentiles, the non-SUV values are the 95^{th} percentiles. Met is the metabolic rate. Pat is the Patlak slope. vb is the vascular fraction.

Dividing patients into two groups according to tumor diagnosis, the SUV time activity curves (TAC's) for both classes are plotted (Figure 44) but reveal no apparent pattern. The two 'other' tumors (myeloid sarcoma and schwannoma) share no common characteristics. One has a pronounced early (vascular) phase and an almost flat late (metabolic) phase, while the other

has initially low values but steadily grows during the entire time interval. The Max, Peak and 98th percentile all show the same order of the patients at 45 minutes. However, some TACs cross each other prior to that. With the mean value the order is slightly altered and the curves for patients with myeloid sarcoma and schwannoma are found more in the middle.

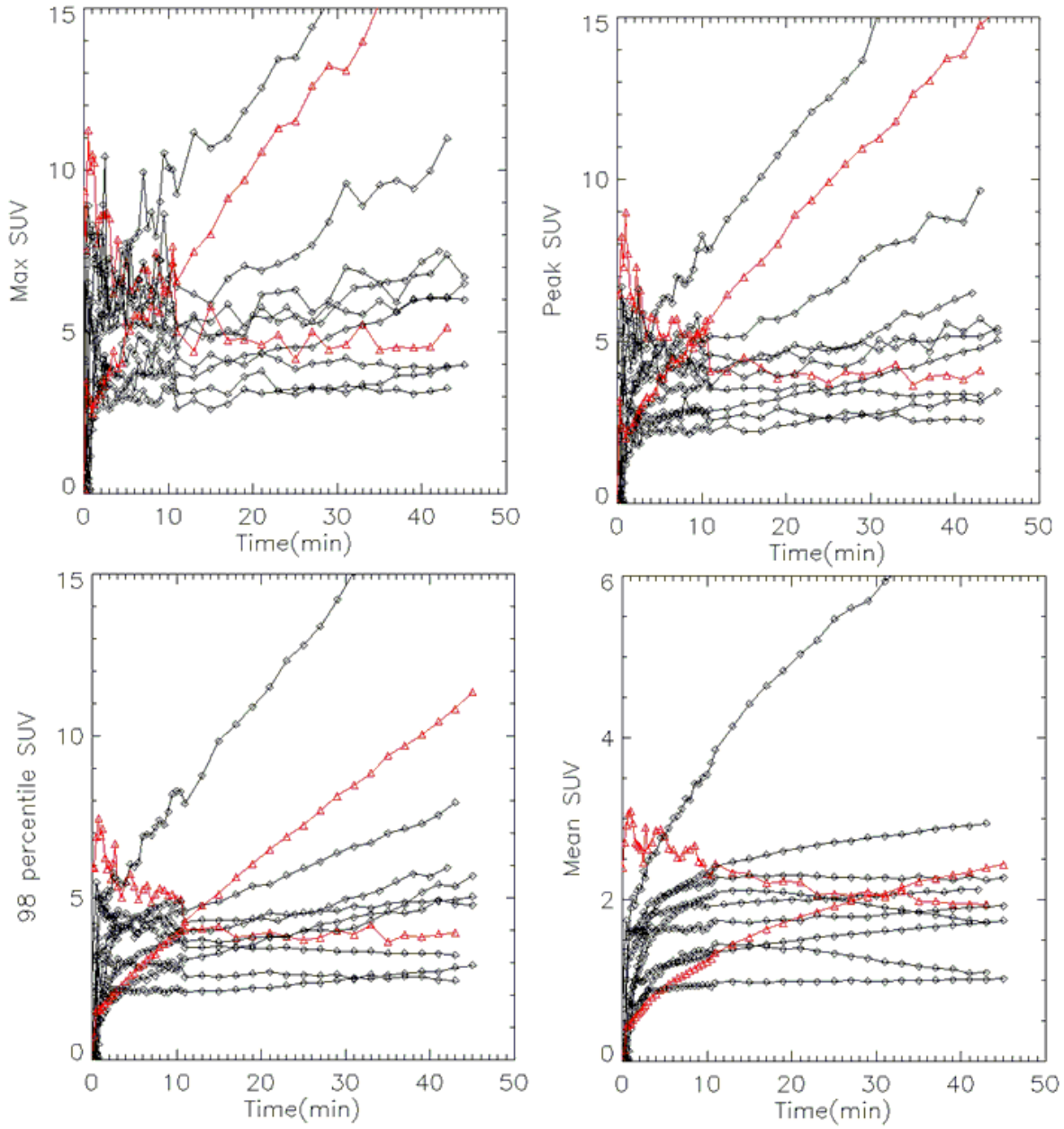

Figure 44. Tumor TAC's of the 11 patients included. 9 with liposarcoma drawn in black and 2 patients termed as 'other' drawn in red. From the top to bottom right, the panels show the Max SUV, the Peak SUV, the 98th percentile SUV and the mean SUV, respectively.

In addition to the tumor values the reference tissue was included (chapter 3.3). The ratio between the SUV value extracted from the tumor and the tissue mean are calculated and

labeled SUV_r. Plotting the SUV_r TAC's for all the patients (Figure 45) reveals that the ratio is greatly amplified in the vascular phase since the reference tissue does not have a high degree of vascularization. Although some alterations from the unscaled versions occurs the general picture concerning the 'other' tumors is maintained.

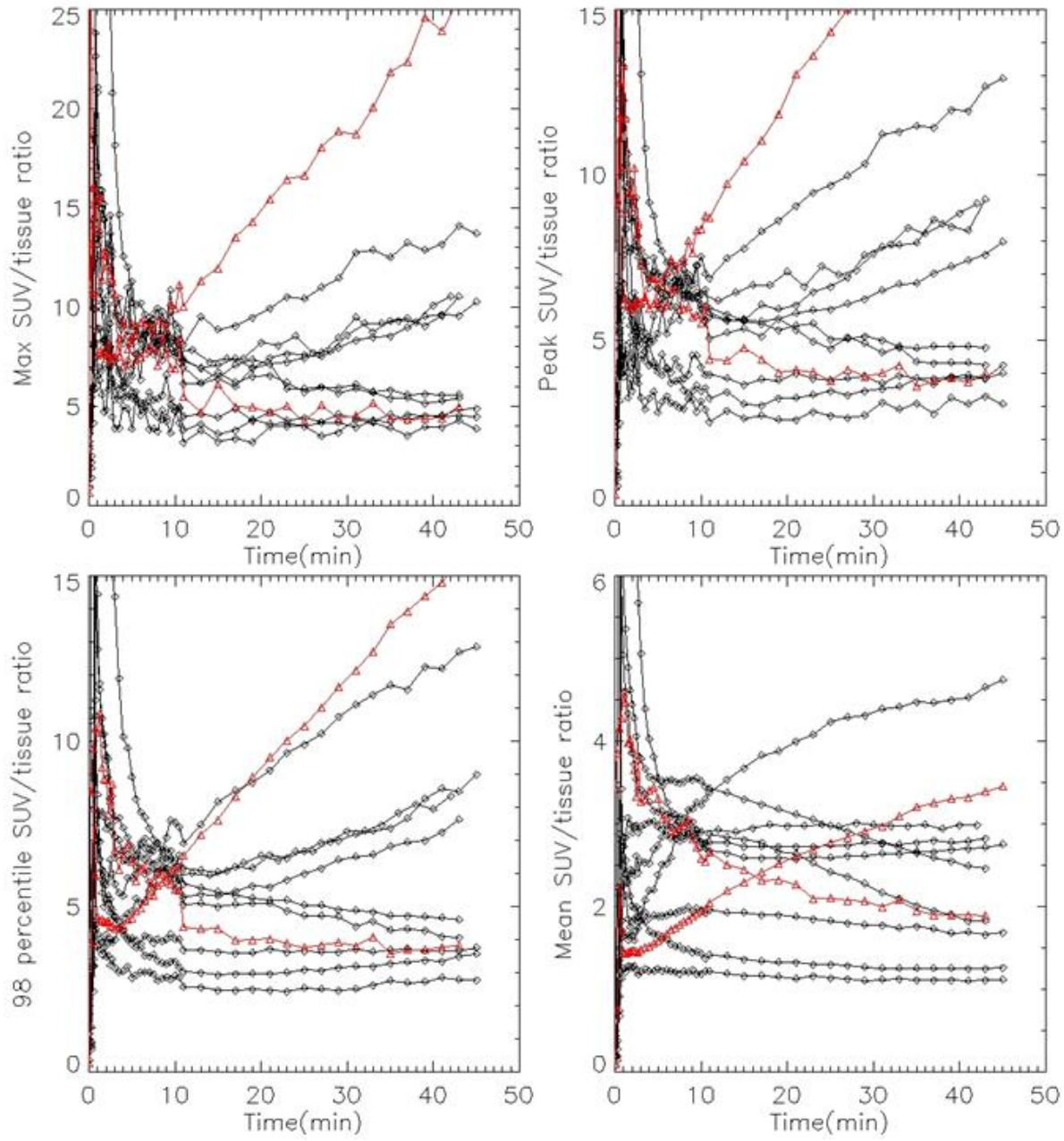

Figure 45. TACs with the 11 patients with liposarcoma have their SUV values scaled to reference tissue and drawn in black. Two patients termed as 'other' have their SUV values scaled and drawn in red. From top left to bottom right, the panels show the Max SUV_r, Peak SUV_r, 98th percentile SUV_r and the mean SUV_r, respectively.

To investigate whether the parameters can detect differences in the classes the Mann-Whitney rank test is used. First, using tumor type as classifier the tumor type classification divides the

groups into 9-2 (9 liposarcomas vs one schwannoma and a myeloid sarcoma) and a U value of 17 is required for statistical significance ($p < 0.05$), see Appendix B. This composition of the U matrix (Table 6) show no indication that any of the parameters have the ability to separate the two classes.

	Peak	max	p98	p95	p90	p70	p50	p30	var	ske	kur
SUV^E	11.00	10.00	10.00	10.00	10.00	9.00	9.00	9.00	10.00	12.00	12.00
SUV_r^L	10.00	11.00	10.00	10.00	10.00	10.00	10.00	11.00	9.00	9.00	11.00
SUV^L	11.00	11.00	11.00	11.00	11.00	10.00	10.00	9.00	10.00	11.00	13.00
SUV_r^E	10.00	12.00	12.00	12.00	11.00	11.00	9.00	10.00	10.00	11.00	13.00
k1	x	14.00	14.00	11.00	11.00	11.00	10.00	10.00	14.00	12.00	11.00
k2	x	14.00	14.00	14.00	13.00	12.00	11.00	9.00	14.00	10.00	10.00
k3	x	12.00	11.00	11.00	11.00	12.00	11.00	10.00	11.00	14.00	15.00
vb	x	15.00	14.00	14.00	13.00	9.00	9.00	9.00	15.00	14.00	11.00
MR_{PET}	x	10.00	10.00	10.00	10.00	10.00	11.00	12.00	9.00	11.00	12.00
Pat_s	x	11.00	11.00	11.00	12.00	12.00	12.00	12.00	11.00	10.00	11.00
Pat_i	x	14.00	10.00	9.00	9.00	11.00	10.00	10.00	12.00	11.00	16.00
r2	x	11.00	10.00	10.00	10.00	10.00	10.00	10.00	13.00	10.00	12.00

Table 6. The Mann-Whitney U values for the division of patients into liposarcoma and leiomyosarcoma in one group (first) and myeloid sarcoma and schwannoma in the other group (second). A blue value indicates that the first group is larger than the second group. Max is the top value, 'p98'-'p30' signifies percentiles, 'var' is the variance, 'ske' is the skewness and 'kur' is the kurtosis of the parameter distribution. SUV is the standard SUV value and SUVr is the tumor/tissue ratio. The color grading indicates a significance level 0.2.

The tumors are classified according to patient age with patients born before 1960 classified into one group and patients born after 1960 in another group. Here there are 4 patients in one class and 7 in the other and to get a significant value the U needs to exceed 24. In this U matrix (Table 7) there are four instances with significant U values. Patlak intercept skewness and the p30-p50-p70 percentiles of the k2 variable.

	Peak	max	p98	p95	p90	p70	p50	p30	var	ske	kur
SUV^E	22.00	23.00	21.00	21.00	21.00	20.00	19.00	18.00	21.00	19.00	19.00
SUV_r^L	19.00	20.00	17.00	17.00	17.00	17.00	17.00	17.00	22.00	15.00	18.00
SUV^L	14.00	15.00	15.00	15.00	15.00	15.00	15.00	14.00	16.00	19.00	19.00
SUV_r^E	22.00	22.00	22.00	22.00	22.00	22.00	22.00	21.00	19.00	17.00	20.00
k1	x	19.00	19.00	18.00	18.00	18.00	17.00	19.00	19.00	17.00	15.00
k2	x	18.00	22.00	23.00	24.00	**26.00**	**27.00**	**27.00**	21.00	23.00	24.00
k3	x	14.00	15.00	15.00	15.00	14.00	16.00	18.00	15.00	18.00	16.00
vb	x	16.00	18.00	17.00	18.00	21.00	21.00	21.00	14.00	18.00	19.00
MR_{PET}	x	18.00	17.00	17.00	17.00	16.00	15.00	15.00	21.00	14.00	19.00
Pat_s	x	18.00	18.00	18.00	17.00	17.00	17.00	16.00	18.00	15.00	15.00
Pat_i	x	24.00	23.00	22.00	22.00	20.00	20.00	19.00	24.00	18.00	19.00
r2	x	22.00	23.00	23.00	23.00	23.00	23.00	23.00	14.00	**26.00**	21.00

Table 7. The Mann-Whitney U values for the 11 patients classified according to age. The first class is patients born before 1960 and the second class is patients born after 1959. Values in boldface are values that indicate statistical significance. Red boxes signify that the second class has the greater values while blue boxes signify that the first class has the greater values. The color grading grades from a significance level 0.2 to 0.02.

The third division is into classes depending on tumor size, greater or less than 200 cm^3. This gives one class with 5 patients and the other with 6 patients and U needs to exceed 26 to be significant. In this U matrix (Table 8) 12 values are indicated as significant. First, the p98-p95-p90 percentiles of the SUV^E and its variance are higher for larger tumors. Second, also the p30-p50-p70-p90 percentiles of the vascular fraction are higher while the kurtosis is lower (small tail). Third, the p98 and p30 percentiles of the K1 parameter are higher for large tumors. Fourth, the variance of the SUV scaled to normal tissue is higher.

The last classification is if the location of the tumor is in an extremity or not. There are 5 in the extremity and 6 in the trunk and the U needs to exceed 26 to implicate significance. The U matrix (Table 9) indicates no significant variables.

	Peak	max	p98	p95	p90	p70	p50	p30	var	ske	kur
SUV^{E}	25.00	26.00	**27.00**	**27.00**	**27.00**	25.00	25.00	25.00	**27.00**	16.00	17.00
SUV_{r}^{L}	17.00	18.00	20.00	20.00	20.00	20.00	20.00	21.00	**27.00**	22.00	20.00
SUV^{L}	20.00	21.00	20.00	19.00	19.00	17.00	19.00	18.00	19.00	16.00	16.00
SUV_{r}^{E}	17.00	16.00	17.00	17.00	18.00	18.00	17.00	18.00	17.00	17.00	17.00
k1	x	24.00	24.00	**27.00**	26.00	26.00	26.00	**27.00**	24.00	16.00	16.00
k2	x	20.00	23.00	23.00	24.00	24.00	25.00	26.00	22.00	16.00	17.00
k3	x	15.00	16.00	16.00	18.00	19.00	20.00	19.00	18.00	22.00	21.00
vb	x	18.00	24.00	26.00	**27.00**	**28.00**	**28.00**	**28.00**	23.00	22.00	**27.00**
MR_{PET}	x	18.00	21.00	21.00	21.00	22.00	23.00	22.00	18.00	15.00	20.00
Pat_{s}	x	19.00	19.00	20.00	21.00	21.00	21.00	22.00	20.00	19.00	19.00
Pat_{i}	x	19.00	19.00	20.00	20.00	20.00	21.00	21.00	18.00	18.00	18.00
r2	x	18.00	22.00	22.00	22.00	22.00	22.00	22.00	17.00	22.00	15.00

Table 8. The Mann-Whitney U values for the 11 patients classified according to tumor size. The first class is large tumors and the second class is small tumors. Values in boldface are values that indicate statistical significance. Red boxes signify that the second class has the greater values while blue boxes signify that the first class has the greater values. The color grading grades from a significance level 0.2 to 0.02.

	Peak	max	p98	p95	p90	p70	p50	p30	var	ske	kur
SUV^{E}	18.00	19.00	19.00	19.00	19.00	19.00	18.00	17.00	19.00	16.00	21.00
SUV_{r}^{L}	24.00	23.00	23.00	23.00	23.00	21.00	21.00	20.00	18.00	20.00	15.00
SUV^{L}	16.00	17.00	19.00	19.00	19.00	16.00	15.00	16.00	20.00	16.00	16.00
SUV_{r}^{E}	20.00	20.00	22.00	22.00	21.00	21.00	20.00	18.00	22.00	17.00	16.00
k1	x	25.00	25.00	23.00	24.00	24.00	23.00	20.00	25.00	21.00	21.00
k2	x	18.00	17.00	17.00	18.00	15.00	16.00	15.00	19.00	19.00	19.00
k3	x	22.00	20.00	20.00	20.00	20.00	20.00	21.00	20.00	18.00	17.00
vb	x	25.00	21.00	19.00	18.00	16.00	17.00	17.00	23.00	21.00	22.00
MR_{PET}	x	22.00	20.00	20.00	20.00	18.00	18.00	15.00	21.00	19.00	19.00
Pat_{s}	x	24.00	21.00	20.00	20.00	20.00	20.00	20.00	20.00	21.00	21.00
Pat_{i}	x	24.00	25.00	25.00	25.00	24.00	23.00	21.00	25.00	17.00	16.00
r2	x	19.00	23.00	23.00	23.00	22.00	22.00	22.00	17.00	21.00	17.00

Table 9. The Mann-Whitney U values for the 11 patients classified according to tumor location. The first class is tumors located in the trunk and the second class is tumors located in the extremity. Blue boxes signify that the first class has the greater values with a significance level 0.2 to 0.1.

4.6 RT branch

Some of the patients where subjected to RT and multiple PET scans were conducted. One scan was conducted prior radiation therapy, one during the first week, one during the last week, and finally one 3 months after the treatment. As given in Table 10, Patient 2 and 5 show little changes from pre- to post radiotherapy, patient 9 have some decrease in SUV^L and patient 10 has some increase in SUV^E and decrease in SUV^L.

	SUV^E	SUV_r^E	SUV^L	SUV_r^L	K1	k2	k3	Met	Pat	VB
Patient 2, sequence 1a	6.0	8.8	5.1	5.0	1.1	1.6	0.02	0.009	0.01	0.21
Patient 2, sequence 1b	7.0	9.1	4.6	5.1	0.5	0.8	0.02	0.009	0.02	0.08
Patient 2, sequence 2	8.8	6.9	5.3	3.7	0.9	1.3	0.01	0.007	0.02	0.11
Patient 2, sequence 3	8.1	6.7	5.3	3.2	1.0	1.3	0.01	0.008	0.01	0.10
Patient 2, sequence 4	8.1	6.6	6.4	4.1	0.9	1.3	0.02	0.011	0.01	0.31
Patient 5, sequence 1	3.2	7.3	7.4	7.8	0.2	0.4	0.07	0.023	0.03	0.11
Patient 5, sequence 2	3.1	7.9	5.9	6.1	0.2	0.4	0.06	0.019	0.02	0.14
Patient 5, sequence 4	3.6	6.0	6.5	7.6	0.3	0.5	0.05	0.030	0.03	0.09
Patient 9, sequence 1	1.9	4.5	10.5	14.8	0.4	1.4	0.22	0.084	0.12	0.19
Patient 9, sequence 2	2.4	9.6	13.8	16.2	0.4	1.8	0.75	0.120	0.24	0.32
Patient 9, sequence 3	1.7	8.3	9.3	15.1	0.4	0.7	0.21	0.084	0.12	0.43
Patient 10, sequence 1	2.1	5.1	2.8	3.4	0.1	0.6	0.06	0.008	0.01	0.10
Patient 10, sequence 2	3.0	7.0	2.6	3.0	0.1	0.3	0.01	0.002	0.01	0.06
Patient 10, sequence 3	3.9	7.8	2.7	2.7	0.2	0.6	0.03	0.005	0.01	0.12
Patient 10, sequence 4	3.5	8.7	2.4	3.3	0.2	0.6	0.02	0.005	0.01	0.08

Table 10. Summary of the RT patient's most important values. SUV^E means the value from the 1:30-2:45 min timeframe. SUV_r^E is the same value normalized to the reference tissue. SUV^L is the SUV value from the 43-45 timeframe and the respective scaled value is the SUV_r^L. All SUV values are the 98th percentiles, the following non SUV values are the 95th percentiles. Met is the metabolic rate. Pat is the Patlak slope. VB is the vascular fraction. Patient 2 has two initial scans about 6 months apart, a and b.

To make a simple evaluation of the development of the tumors during radiation therapy several values are plotted as a function of the therapy progression. The SUV^E 98th percentile

shows a weakly increasing curve while the corresponding SUV_r^E show a flat curve (Figure 46). Patient 5 has a SUV^E about twice that of the other patients while the SUV_r^E is similar in all patients.

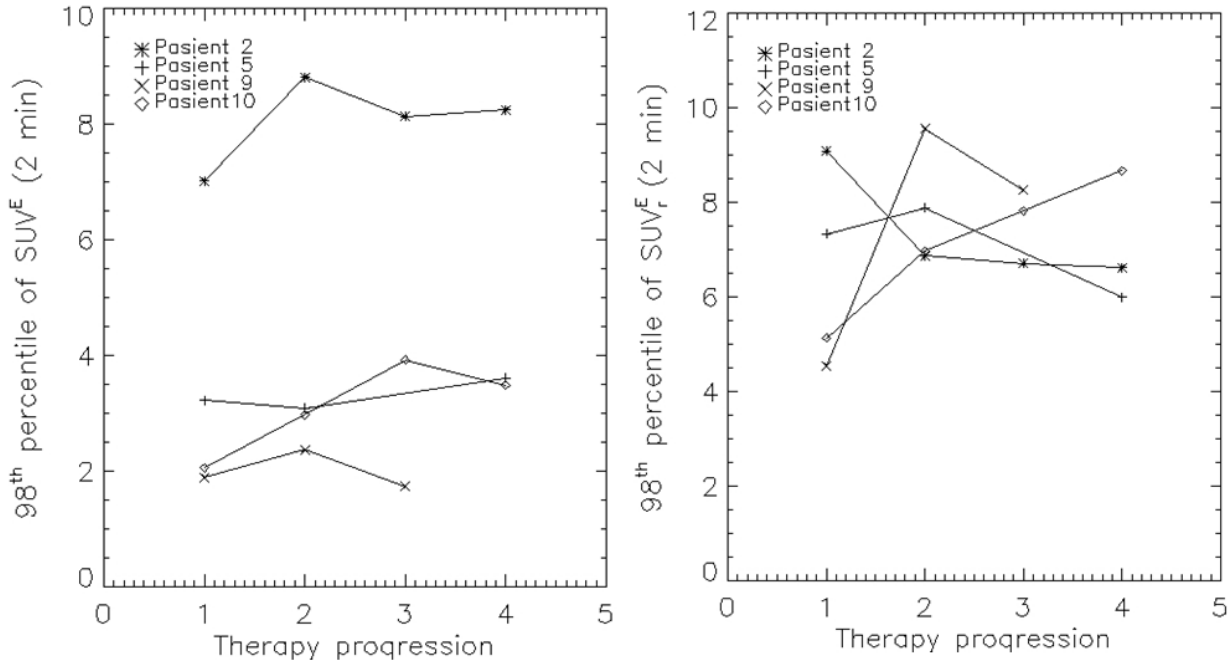

Figure 46. Plots of the 98th percentile of the SUV at 2 minutes as a function of radiation progression. In the left plot the SUV value is used directly while in the right plot the SUV is scaled to the mean of the reference tissue. The first stage (1) is just before treatment. The second stage (2) is during the first week of treatment. The third stage (3) is during the last week of therapy. The fourth stage is about 3 months after treatment.

The SUV^L 98th percentile show a flat curve and so does the SUV_r^L (Figure 47). Patient 9 is in both instances separated by the other curves with about a factor of two but in the SUV_r^L case the difference is slightly greater. The lowest values are found in patient 10 but the difference is slightly less in the SUV_r^L case.

The 98th percentile of the K1 is increasing, with about a magnitude of 2, during therapy for all patients except patient 9 which has a slightly decreasing trend (Figure 48). Patient 5 has the highest values around 0.8 while patient 9 have values about half of that and patients 2 and 10 have values about a quarter of the values of patient 5. The development of the vascular fraction is flat for all the patients (Figure 48). All patients have values in the 0.01-0.03 range except patient 9 which have a value about ten times higher.

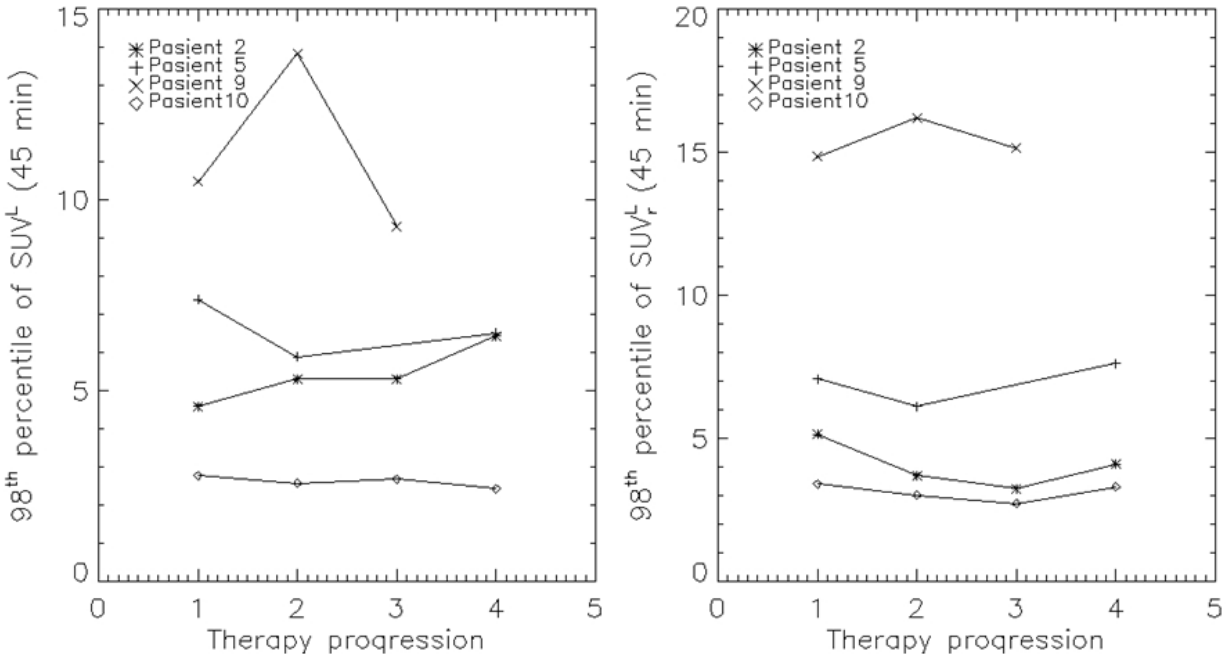

Figure 47. Plots of the 98th percentile of the SUV at 45 minutes as a function of radiation progression. In the left plot the SUV value is used directly while in the right plot the SUV is scaled to the median of the reference tissue. The progressions are the same as in the previous figure (Figure 46)

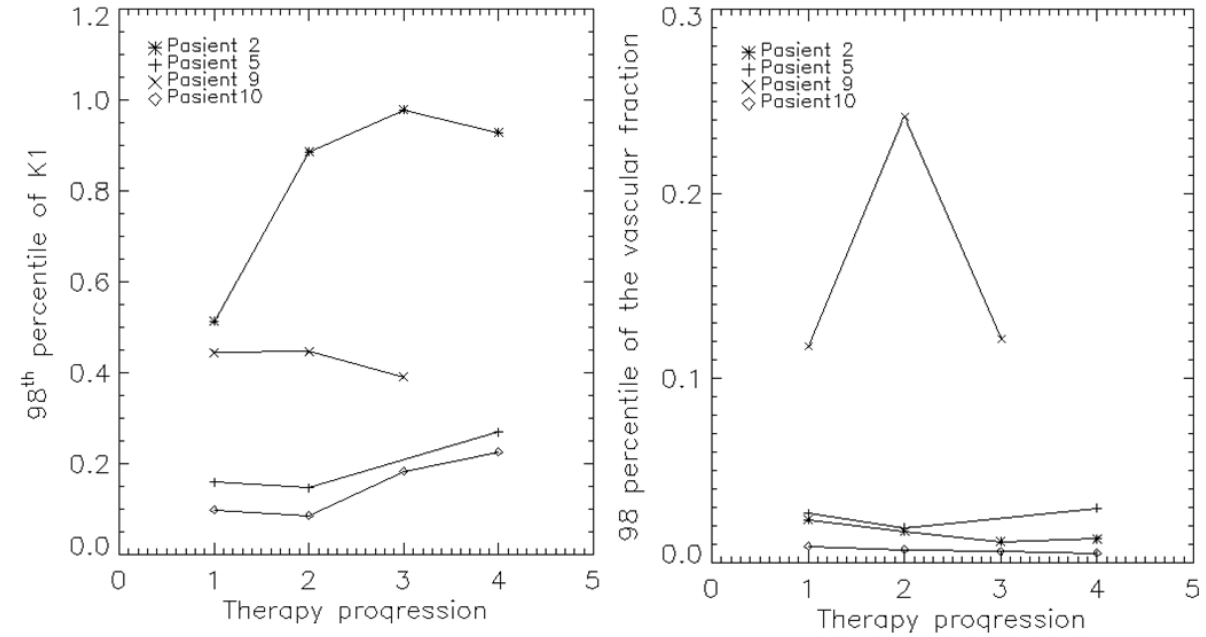

Figure 48. Plots of the 98th percentile of K1 and vb as a function of therapy progression. To the left the K1 is plotted and to the right the vascular fraction (vb) is plotted. The progression is the same as in the previous figure (Figure 46).

The metabolic rate show a flat development (Figure 49) with patient 9 having the highest values (~0.1), patient 2 having values about a third of patient 9 (~0.03) and patients 5 and 10 having values about a third of that again (~0.1). The heterogeneity is measured through the

standard deviation of the SUV^L normalized against the SUV^L Peak value (Figure 49). It is a rather flat curve for patient 2 a slightly decreasing curve for patient 10, it is decreasing for patient 5 while increasing for patient 9.

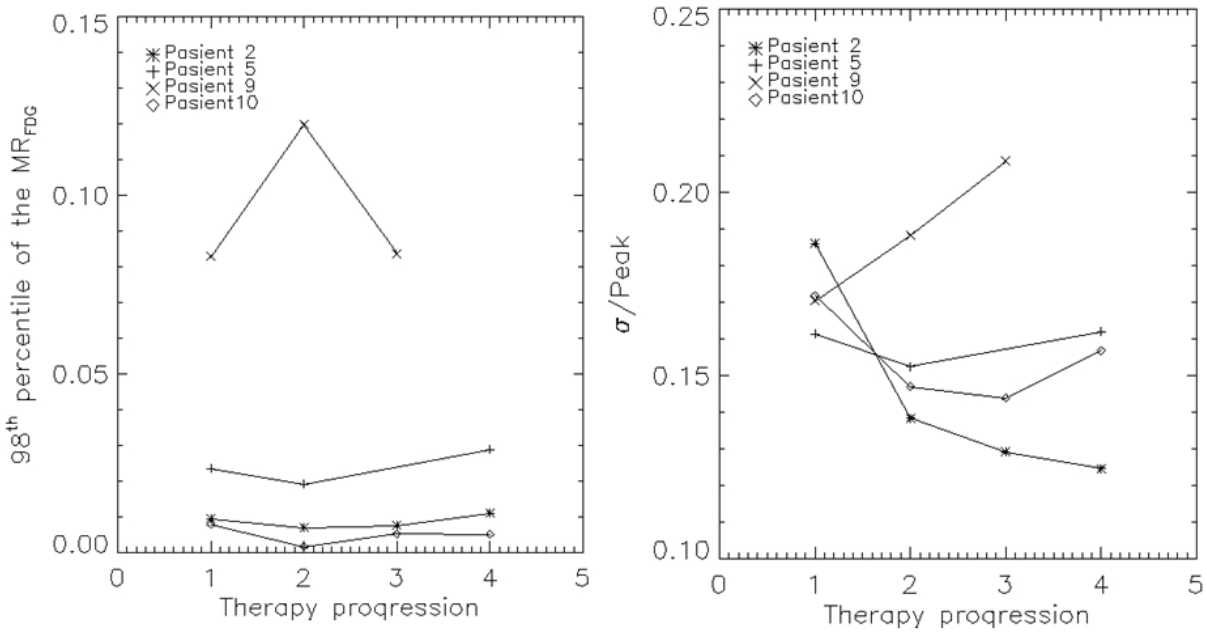

Figure 49. The right image is a plot of the 98th percentile of the metabolic rate. The left image is a plot of the standard deviation of the tumor normalized to the Peak value. The progression is the same as in the previous figure (Figure 46).

5 Discussion

5.1 Assumptions and limitations

In the current work twelve parameters describing vasculature and metabolism have been examined to see if they could predict biological features in patients. The variables are partly derived directly from the PET images, partly from a Patlak graph analysis and partly from a two compartment model. A shared problem with both Patlak analysis and the pharmacokinetic variables is that they rely on the plasma function. Since this plasma function is obtained non-invasively from the PET images certain assumptions are made. The metabolism of red blood cells is assumed negligible, the FDG binding to red blood cells and other proteins is assumed to be short and the exchange between plasma and red blood cells is assumed to be fast. In this way an image based plasma function is used for sampling from the whole blood. Although several adjustments have been made to get a good function it is greatly dependent on the timing when the injection (the bolus) reach the artery from which we sample the plasma function. Patient 5 illustrate this with two scans, one early and one late in treatment, the plasma has a SUV peak of 10 in one case and 20 in the other (Figure 50).

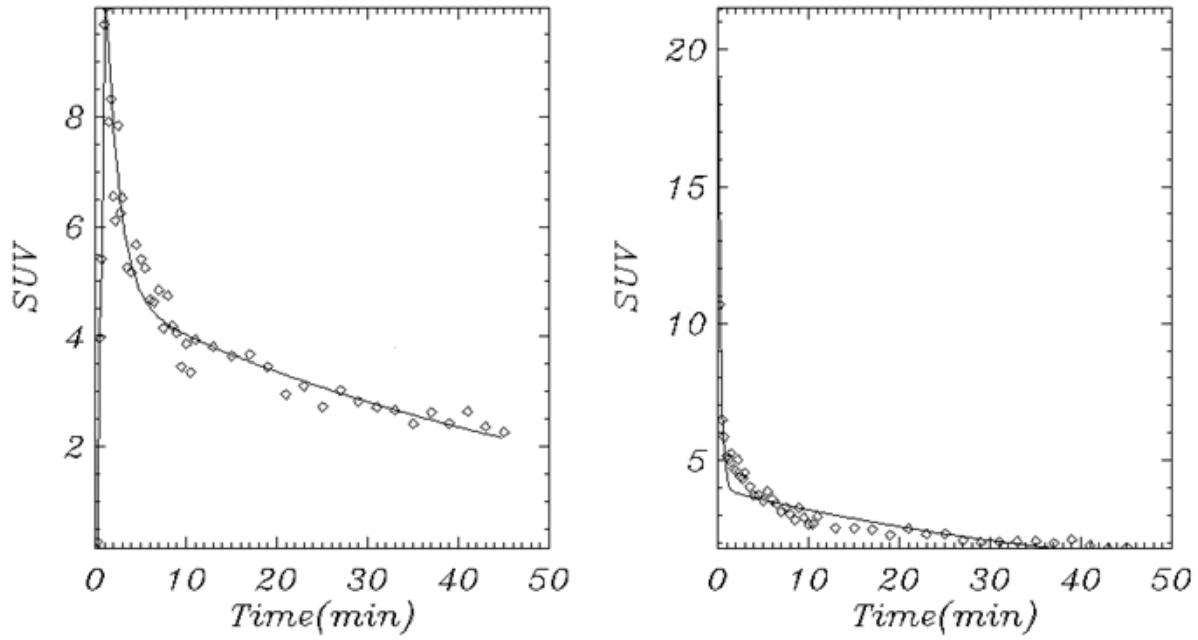

Figure 50. The plasma function of patient 5. To the left is the plasma function obtained during the first week of radiation therapy. To the right is the plasma function obtained during the last week of radiation therapy. The diamonds are the original data points and the solid line is the fitted biexponential function.

This vulnerability of the initial peak is linked to that the time resolution is long (15 seconds) compared to the bolus development (one heartbeat about every second). This means that the bolus amplitude recorded depend on if the bolus reach the sampling voxel at the beginning of a sample period or during the late part of the sample period. The first peak is the critical point in the curve fitting as the bolus concentration is at its highest directly after injection. Thus, it is desirable to have as short sample intervals as possible in the initial time segment. On the other hand, the statistical noise in PET scanning increases with the time resolution, and thus plays an important role during the initial part of the dynamic scan compared to the late points. To evaluate these two conflicting elements the signal to noise ratio, SNR, can be used. The peak plasma seems to be around 12 SUV while the late signal is around 2 SUV, giving a 6 to 1 ratio. Since the noise in the image is proportional to the square root of detections, it is an average of all coincidences detected within the time interval. It would seem possible to reduce the sampling time to 5s to capture the initial peak with a comparable SNR. This would however have a practical consequence as the computer program loads the arrays into the computer memory as a contiguous segment. Since the system running the software is a 32 bit Windows XP it has a fundamental memory limit of 2GB. Producing three times as many initial images would require a rework of the software memory handling or a switch to a higher capacity system. Thus the sampling frequency used in the current work is a compromise between the number of points available in the early phase and the late phase. Most points are found in the early phase, however, since the time interval within the bolus can occur is rather large (1-3 minutes), 15 seconds are deemed adequate.

Another issue with the plasma sampling is the partial volume effect. The aorta is the largest artery of the body with a starting diameter of 20-30mm and from the central part of the aorta several sample points can be used. The four principal divisions of the aorta are the ascending aorta, the arch of aorta, the thoracic aorta and the abdominal aorta (Appendix A) and as the aorta branches off it gradually decreases in size. When the iliac and femoral parts are reached the diameter has decreased to about 5 mm and the later branch into tibial and fibular arteries further reduces this. 5mm is the dimensions of the PET voxels and as explained in chapter 3.4 a cylinder has to be 10-15 mm in diameter to be correctly represented at that resolution. This means that for the tumors in the lower legs it is inevitable that partial volume effects are introduced. The exclusion by quality method described in chapter 3.4 is used to correct for partial volume effects. However this method only partially corrects for partial volume effects

in arteries of a diameter less than 7 mm and it also limits the number of sampling points, increasing the effect of noise.

As described in chapter 2.3 the pharmacokinetic model itself has a long list of assumptions. The most central assumption is that FDG should be evenly distributed in the compartments. This in turn requires good diffusion, enough glut transporters and short FDG-protein binding times. In the blood there is a gradient of glucose in the arterial-venous direction that is assumed to be negligible and in the tissue it is assumed to be no difference in FDG in the interstitial space and in free (unbound) FDG found intracellularly. The signal detected by the PET scanner is of the FDG, not the actual glucose, and though they behave in a similar way they are not identical; glut transporters prefer deoxyglucose while hexokinase prefer glucose. Furthermore the transfer rates are assumed to be linear (that is load independent) though biological systems are in their nature adaptive. We are recording the signal in an atypical setting, the patient does not usually fast, and as such we record the fasted metabolism not the normal metabolism. The perfusion over the capillary walls is assumed to be only dependent on blood flow though tumors have chaotic blood flow and low perfusion. The number of assumptions are required however to reduce the problem complexity sufficiently, not only from a calculation point of view, but primarily to create a model that actually reflects the underlying biology. A complex model has a tendency to perfectly describe the data set used to train the algorithm, but usually it leads to over-fitting and low robustness when real, imprecise and noisy data are presented.

11 patients are from a statistical point of view limited to examining a single variable if results of a statistical significance are to be found (Chapter 3.7). Furthermore as the number of variables tested for are 12, with more than one aspect of the variables examined, random variations showing up as correlations are assumed to occur naturally. 9 of the 11 patients are diagnosed with liposarcoma and classification is attempted with these as one class and the other two as the other class. A fundamental problem with this division is that the heterogeneity of liposarcomas, and it is reported in the literature that they are a difficult entity on which to perform clinical studies (Matushansky and Maki 2005). The last two patients, one with a schwannoma and one with a myeloid sarcoma, are also dissimilar. Schwannoma is a cancer connected to peripheral nerves while myeloid sarcoma is a special type of leukemia expressed as a solid tumor outside the bone marrow. Since no information on recurrence free survival rate was available alternative divisions like patient age, tumor size and tumor

localization are attempted to simulate variations in circulation though their clinical use is limited.

5.2 Model robustness

To quantify the fitting quality of our model the squared Pearson correlation coefficient, r^2, is calculated for the original data and the fitted curve (Figure 51). The mean of each patient's tumor voxels indicate that patient 1, 2 and 4 has the poorest fit with an r^2 mean in the range 0.1-0.2, patients 3, 8 and 12 have intermediate r^2 values of about 0.5 while patients 5,6,7,9 and 10 have r^2 values in the range 0.7-0.8. In chapter 3.5 it was described how a spatial smoothing filter was applied to the input values. The effect of this smoothing is a slightly decreased (about 5%) r^2 of the curve fit, when comparing the raw unsmoothed data with the fit. However it is assumed that rapid local changes of the parameters are not consistent with the biological nature of the processes described and that the smoothed fit is a better description of the true underlying biological processes. Calculating the r^2 between the smoothed fit and the smoothed data display an increase of 0.2 for most patients. The smoothing reduces the effect of extreme outliers but introduce a subjective component as curve fitting is no longer after the raw data but rather after data that are perceived as more correct.

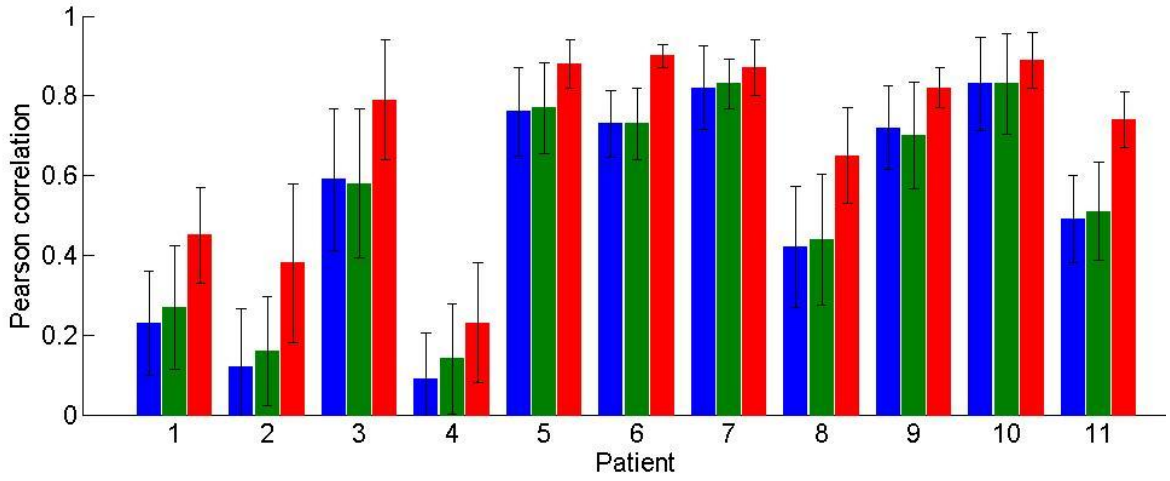

Figure 51. Graphs of the curve fitting quality (r^2). The green bars are between PET data and PET data fit. The blue bars are between the PET data and the smoothed PET data fit. The red bar is between the smoothed PET data and the smoothed PET data fit.

To search for a systemic bias of the model the original data minus the fitted data, the residual, is calculated for each timeframe. The residual of every voxel in all the patients are added together for every time sample point and the mean over the patients are calculated (Figure 52). If the errors in the fitting are random they should even out and look like white noise around zero. The residuals of the patients in this case are rather well centered on zero, though they are not evenly distributed. The plasma function is underestimated initially, overestimated in the mid range, and underestimated in the last part. This stems from that the biexponential function contains a rather sharp corner when the data does not resemble an exponential (Figure 50). The less the data points resemble an actual exponential the more distinct this corner becomes. The error is in the range ±0.2 SUV (~10%) except during the initial peak that reaches an overestimation of 0.5 SUV (~5%). The residual plot of the tumor region show that the model consistently over estimates the function by about 0.07 SUV, except in the initial phase where it underestimates it. The residual has a rather sharp peak at early time points with a minimum of -0.4 SUV.

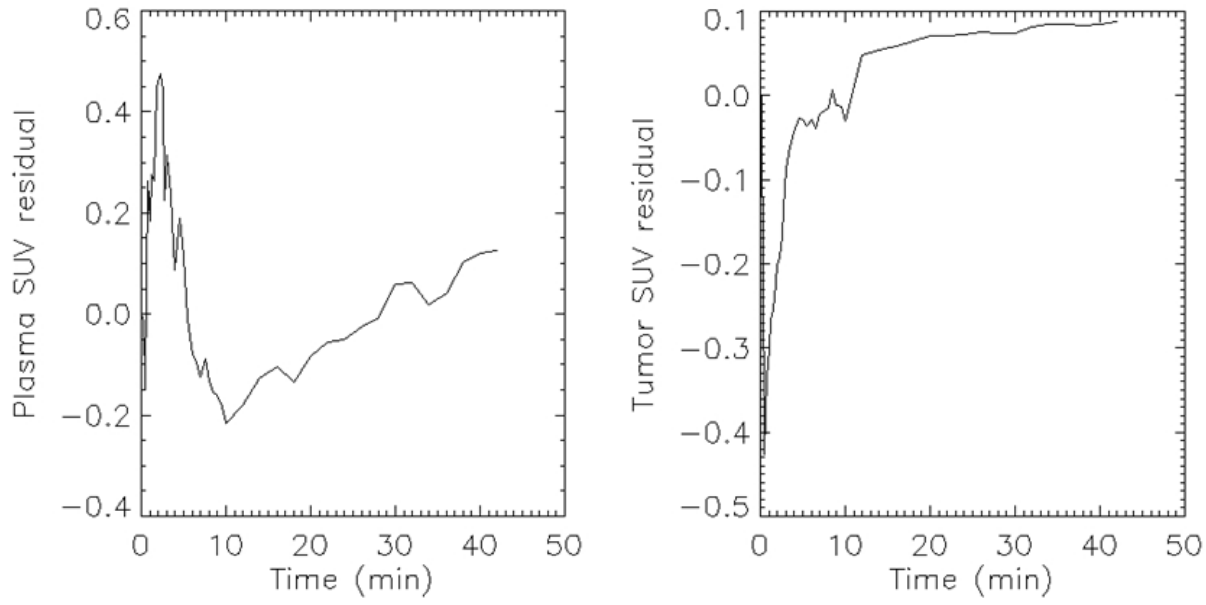

Figure 52. The combined residuals from the voxelwise model fitting of patients 1-11. To the left the combined plasma residuals are plotted. To the right the combined tumor residuals are plotted.

The residuals are rather small compared to the actual signal, but indicate areas for possible improvements in the modeling. Most likely the peak stems form the initial assumption that the concentration is evenly distributed in the blood. Directly after injection the FDG is concentrated in a bolus and the plasma sample position is not the same as the tumor position. This introduces a time delay; the model assumes that the bolus begins when it reaches the

plasma sampling voxels while in reality it begins when it reaches the tumor. This way the model think the curve should begin earlier than it actually does and the result is an initial overestimation.

A potential issue with the model is the numeric nature of the solution. The optimization of the least square differential is an iterative process and it is possible that the algorithm is trapped in a local minimum before it reaches the global minimum. This means that the result is dependent on the initial values set by the user. However, as setting the initial values for each voxel in the tumor is not feasible a global starting value is used for all voxels and all patients. To examine the possibility of this corrupting the result all values are initially calculated for the original setting of initial values. Then the initial values of K1, k2, k3 and step size are increased by 100% or reduced by 50%. The difference between the voxel values derived from the original settings and the voxels of the 80 permutations are calculated and from this the maximum difference, the mean value and standard deviation is extracted (Table 11).

Most patients are invariant to the initial values but for patient 3, 4, 7 and 9 a slight effect can be detected. The max difference can be comparable to the calculated values indicating a voxel completely reversing its value. However the standard deviation indicates that the average change is relatively small and the majority of the points undergo much smaller alterations. Patient 4 seems to have the most unstable fitting with a standard deviation of 0.04 for K1, 0.3 for k2, 0.001 for k3 and 0.003 for vb. The mean values are found around zero which is expected as the effect of the changed initial values should be random without a bias towards increasing or decreasing values.

As mentioned earlier in the discussion noise in the plasma function has a fundamental effect on all the calculations. This noise arises primarily due to the timing of the bolus and motion artifacts; however other types of noise also affect the calculations. There are always statistical variations in how many decays occurring in a volume element, how many of the resulting photons are detected by the scanner, how much scatter there is and how much random background noise is detected. As mentioned in chapter 3.4 and 4.3 this general white noise can also be amplified by the curve fitting and a smoothing filter was thus used to limit this. To model the effect of noise it is desired to recreate the input PET array but with another noise image. Adding white Gaussian noise of the same magnitude as present in the image does this but also amplifies the noise level by about 40%.

Patient	$K1 \cdot 10^{-3}$			$k2 \cdot 10^{-3}$			$k3 \cdot 10^{-4}$			$vb \cdot 10^{-4}$		
	Max	Mean	Std	Max	Mean	Std	Max	Mean	Std	Max	Mean	Std
1	0.11	0.00	0.02	0.36	0.00	0.03	004	0.00	0.00	0.03	0.00	0.01
2	0.60	0.01	0.07	1.18	0.02	0.13	0.04	0.00	0.00	0.06	0.00	0.03
3	94	0.02	2.7	98	0.04	2.79	35	0.00	1.09	986	-0.72	43
4	315	-1.08	44	157	-0.64	29	75	0.00	8.96	1803	4.08	251
5	0.03	0.00	0.00	0.10	0.00	0.01	0.08	0.00	0.01	0.12	-0.01	0.02
6	0.20	0.00	0.57	0.60	0.04	0.23	0.03	0.00	0.01	79	0.11	3.7
7	727	0.89	26	913	1.11	35	63	0.03	2.73	779	-1.19	44.9
8	0.05	0.00	0.01	0.05	0.01	0.01	0.09	0.00	0.02	0.27	-0.02	0.05
9	47	0.03	1.04	27	0.03	1.00	174	0.05	25	656	-0.42	23
10	0.05	0.00	0.00	0.57	0.00	0.04	0.07	0.00	0.01	0.02	0.00	0.01
11	0.11	0.00	0.01	4.45	0.02	0.19	1.27	0.01	0.05	0.04	0.00	0.01

Table 11. The max value, mean value and standard deviations of the difference introduced into patients 1-11 by increasing the initial values (K1, k2, k3, and step size) used by the least square fitting by 100% or reducing them by 50%.

To measure white noise in an image the standard deviation of a dark area may be used. However, since the noise in a PET scanner is not uniform and the dark areas are located in the edges of the image while the tissue is in the center an alternative method is used. By applying a smoothing filter to the reference tissue an image of the low frequencies is obtained. Subtracting this from the original image leaves an image with the high frequencies intact while the low frequencies are reduced. Extracting the standard deviation from this image produce an estimate of the noise present in the image and adding Gaussian noise with this standard deviation to the original image creates a noise distorted image. Then subtracting the values computed from the distorted image from the values computed from the undisturbed original image gives a difference array. From this array the max difference, the mean difference and the standard deviation of the difference is extracted (Table 12).

Patient	$K1 \cdot 10^{-3}$			$k2 \cdot 10^{-3}$			$k3 \cdot 10^{-4}$			$vb \cdot 10^{-4}$		
	Max	Mean	Std	Max	Mean	Std	Max	Mean	Std	Max	Mean	Std
1	57	-0.06	1.38	133	-0.11	2.68	44	0.00	0.31	93	-0.11	2.73
2	56	-0.02	0.76	197	-0.04	1.72	53	0.00	0.15	90	-0.04	1.50
3	35	-0.02	0.54	229	-0.07	1.62	212	0.04	0.99	484	-0.10	2.11
4	289	-0.04	1.78	198	-0.07	1.43	24	0.02	0.34	810	-0.32	16.6
5	22	-0.05	0.66	98	-0.17	2.32	185	0.09	1.83	83	-0.20	2.93
6	25	0.00	0.28	78	-0.01	0.62	24	0.00	0.12	428	0.00	0.67
7	944	0.58	13.0	935	0.80	16.8	215	0.28	5.57	685	-0.23	6.84
8	241	0.08	3.04	320	0.12	4.42	570	0.23	8.61	878	-0.08	4.48
9	244	-0.09	1.51	1920	-0.62	12.2	2450	0.22	10.6	627	-0.04	4.50
10	89	-0.18	2.78	2190	-3.56	58.3	995	-0.81	13.9	615	-1.83	27.0
11	67	-0.03	0.78	488	-0.13	4.8	365	0.04	2.04	80	-0.08	2.13

Table 12. The max value, mean value and standard deviations of the difference introduced into patients 1-11 by adding Gaussian noise with the standard deviation of the reference tissue into the image data used by the two compartment model.

The effects of the added noise is most pronounced in patient 7 and 10 where some voxels differences are as great as the voxel values however the standard deviation is low indicating that in most cases the voxel values are stable. It would seem that the software is rather resistant to noise in the input parameters and values. However since certain voxels does report a significant change it should be expected that some noise amplified outliers can occur. As such taking the max or min value of the tumor region is not recommended as this max value could be one such erroneous result of the curve fitting. A percentile like the 90^{th}percentile however would require many points to be affected, this is unlikely however as it would have been indicated by a higher standard deviation.

5.3 The value of dynamic PET scanning

Classification according to tumor type and tumor location showed no correlation. The best descriptor for age seems to be the k2 parameter which has more low valued voxels in older patients. This result indicates that the wash-in is unrelated to age while the wash-out is lowered in certain tumor regions of elderly patients. The k2 variable defines the level of concentration in tissue during 'equilibrium' and the low values might indicate low circulation (FDG is trapped passively for a long time in the tissue). However the other vascular parameters do not show any correlation with age and as such it is reasonable to see the result as a chance finding. For classification according to tumor size the early SUV has higher values in large tumors. The vb shows fewer low values in large tumors and K1 has higher values in large tumors. This might indicate that large tumors have a higher wash-in with high K1, high early SUV and higher mean vb. The early SUV, K1 and vb are linked (Table 4) and their combined correlated result might indicate an actual effect. For most patients the tumor–to-tissue rate show a higher contrast between the early and late SUV (Table 5 and Table 10), however, it does not seem that this translates into an improvement in the prediction rate (Table 6-Table 9). It seems that dynamic PET can detect an abnormal vasculature and that K1, k2 and vb describe different aspects of the vasculature; however it was not found to impart an ability to classify liposarcomas/leiomyosarcomas nor was it determined if this description was superior to using the SUV value 2 minutes after injection.

When it comes to parameter selection it would seem that the tumor–to-tissue ratio did not provide any improved diagnostic value. The best metabolic value seems to be either the late SUV for its simplicity and widespread use or the metabolic rate for its superior detection of patient 2's tumor. Both values seem robust enough to allow the use of the Max or Peak values. The Patlak values, though mimicking the metabolic rate and vascular fraction, produced images with more noise and no additional information.

The k3 parameter which measure the activity of hexokinase should be linked to malignancy however it may seem that the curve fitting produce false high values. An insignificant voxel with a weak signal can easily have an extremely high k value as long as the free compartment is small and the algorithm is fitting the adapted curve to values dominated by noise. The correlation matrix (Table 4) show that although k3 correlates with MR_{FDG} it does not correlate with the other metabolic parameters (that MR_{FDG} do correlate with). It would seem that in

some instances high k3 values are produced in areas with low circulation which should indicate hypoxia and malignancy, unfortunately in such regions the k3 are easily affected by noise and might not be reliable. It does however add additional information; the tumors in patient 7 and 8 are the two tumors that accumulate the most FDG (Table 5). They also have the two highest MR_{FDG} values and patient 8 has the highest k3 value. Patient 7 on the other hand has a k3 value of less than a third of patient 8 while at the same time patient 7 has a final SUV value 70% greater. This is the result of K1 and k2 being equal resulting in a large free compartment supplying the transfer into the bound compartment.

The metabolic rate is a combination of parameters and demands a well filled free compartment. This leads to a curve fitting with the intended features, which in turn make the metabolic rate robust. An interesting pattern is seen in some tumors (Figure 28) where there is a somewhat anti symmetric relationship between the vascular fraction and MR_{FDG}. It has been reported in the literature that FDG correlates positively with hypoxia and negatively with blood flow and cell proliferation (Pugachev, Ruan et al. 2005) so a hyper metabolic area with somewhat reduced blood flow is not entirely unexpected. Feasibility studies of dynamic FDP PET used in therapy monitoring has been conducted and it has been reported that the best prediction is found in a multiparameter approach combining the mean SUV^L and MR_{FDG} (Dimitrakopoulou-Strauss, Strauss et al. 2010). This is consistent with the observation that patient 2 has a better tumor description than SUV^L (Figure 34) while in general SUV^L works equally well and in patient 4 arguably better (Figure 36).

The description of the vasculature was done by the SUV^E, K1 and vb parameters though they seem to describe different aspects as they are not always correlated with each other. K1 and SUV^E on the other hand seem closely linked. Visually SUV^E seems to have higher contrast. For classifying tumor according to size it seems to be a more robust value, as significant values for p98,p95 and p90 was found. The max value was not significant however indicating that it might be affected by outliers. Outliers were also demonstrated for K1 and k2 in chapter 4.3 and the parameters should only be used with percentiles. Both high and low parameters seem to contain information.

It has been reported in the literature that a 40% decline in SUV_{Max} during therapy indicates a good prognosis for soft tissue sarcomas (Schuetze, Rubin et al. 2005) and that the number of responders are about 35% of the patients (Weber 2005). The data from the radiation therapy branch indicate that none of the patients report a 40% decrease in late SUV and as such none

can be termed responders. The validity of this statement cannot be tested as recurrence free survival for the patients are not available, though the number of patients in itself discourages any concrete conclusion. Though the time trends in SUV for the patients have to be termed flat two notable changes occur. First K1 seems to increase during therapy for 3 out of 4 patients, which may indicate an inflammatory response. Second, the heterogeneity has a slight decrease for patient 5 which is consistent with a decrease in SUV for the top values, tumor cells dying leading to increased necrosis, and an increase of the low values through inflammation. One possible shortcoming of the heterogeneity measure is that it does not take into account the shape of the tumor. It has been reported in the literature that using a geometric primitive such as an ellipsoid distribution and fitting this to the tumor image approximates the background signal. Subtracting this from the image before conducting the heterogeneity calculation gives a measure that can predict patient outcome (Eary, O'Sullivan et al. 2008).

In conclusion it can be said that the implemented software indicated an ability to detect vasculature and metabolic rate though the limited number of patients meant that statistical significance could not be tested for. The best descriptor for metabolic rate was found to be a combination of SUV^{L} and $MR_{FDG.}$ For describing perfusion and vasculature SUV^{E}, K1 and vb seem to describe different aspects of the vasculature. Visually SUV^{E} has the best contrast in the tumor (Figure 19) and its ability to classify large tumors stems from its high values (Table 8). The vb on the other hand has a lower contrast in the tumor (Figure 28), however, its ability to classify large tumors stems from a wider range of values around the median (Table 8). In other words the vb detection is more robust and focuses a general vascularization, and the median value should be used for classification. The SUV^{E} on the other hand illustrates areas of great leakiness. In SUV^{E} significant correlation with tumor size was found for all high percentiles, but not for the Max value. This may indicate that the Max value is disturbed by an outlier, and it is not recommended to use the max value, but rather use the 95^{th} percentile. K1 showed the poorest correlation with tumor size of the three (Table 8) and a visual contrast similar to vb (Figure 25). In the RT patients K1 showed the best indication of inflammatory response (Figure 48), however, Table 11 and Table 12 indicate that computationally introduced outliers can occur and it is recommended to use the 95^{th} percentile with K1. The attempts of classification where not conducted on clinical values, such as tumor type, aggressiveness or recurrence free survival, and it cannot be concluded that the additional time in the scanner is warranted. However, additional information compared to a standard PET

scan was found in the pharmacokinetic parameters and it seems that a further investigation into if these parameters can be linked to clinical values seems justified.

5.4 Further work

The make the findings from the classification clinically relevant they should be linked to prognosis. Liposarcomas are a heterogeneous group of tumors but the use of histological data would enable us to subdivide the tumors into smaller groups that also would contain information on the tumor prognosis. One relevant division is between the well-differentiated liposarcoma, a usually benign tumor and the dedifferentiated liposarcoma, a usually malign tumor. The well-differentiated liposarcoma can be distinguished from the dedifferentiated liposarcoma histologicaly by identification of fibrotic regions. Another natural division is between the myxoid liposarcoma, a usually benign tumor and the round cell liposarcoma, a usually malign tumor. With respect to PET, the image matrix of the SUV voxels can produce user defined 2D slices of any orientation. In a resected tumor, regions of necrosis, fibrosis or regions containing large fractions of round cells can be identified and correlated to an image of the calculated parameters. The resolution is low however so texture analysis is limited and the analysis is expected to be confined to large tumors.

As shown in chapter 5.2 there is a certain model bias indicating that the plasma might require a more flexible curve fitting to get evenly distributed residuals. The fitting is performed to remove noise in the plasma function but a further investigation into other functions such as a spline fitting might give a better results. Another area for possible improvement is indicated by the tumor residual's distinct initial dip. Since it is most likely connected to a time delay between the plasma sample area and the tumor area, it should be possible to correct for it by detecting the initial peak of the plasma function and the initial peak inside the tumor. The dime difference between them should constitute the delay and shifting the plasma input curve by this time segment should correct it. However the time delay is probably less than 15 seconds and to detect the peaks properly a higher time sample rate is required. An alternative method would be to enter the time difference into the curve fitting algorithm as a fifth variable and calculate the optimum time difference according to the least squared fitting.

When choosing the time interval for describing the vasculature, earlier time points gives better contrast. But during the first minute the changes are abrupt and the timing of the

injection itself become critical. Variations of about 15 seconds occur in injection timing, and different injection points are used resulting in another ~30 second's variation in the time it takes for the bolus to reach the tumor. Rather than using the average over a fixed time interval to evaluate the vascularization/perfusion of the tumor it could be improved by higher sampling frequency and an automatic detection of the initial bolus. This way, though the curve fitting algorithm might use longer sampling intervals to reduce noise, the first sampling interval could be set to always start when the bolus reaches the voxel. Another advantage of increased sampling rate is that it has been argued that with a sampling time as low as 2 seconds the tracer's initial movements through the capillary bed may be described (Tofts, Brix et al. 1999). At such a sampling rate it would be possible to observe the wash-in before the wash-out comes into play thereby simplifying the measurement of the perfusion itself. However at such a high sampling frequency the noise component may be too high. If such an approach is attempted it should probably be limited to tumors in the thorax as the bolus is required to reach the tumor in a concentrated form to keep the SNR high.

References

Alberts, B., A. Johnson, et al. (2008). Molecular biology of the cell. New York, Garland Science.

Avery, L. (2002). "Elementary Statistics for Biologists." from http://elegans.swmed.edu/~leon/core_2002/stats/formulas.htm#utest.

Bastiaannet, E., H. Groen, et al. (2004). "The value of FDG-PET in the detection, grading and response to therapy of soft tissue and bone sarcomas; a systematic review and meta-analysis." Cancer treatment reviews **30**(1): 83-101.

Benz, M. R., V. Evilevitch, et al. (2008). "Treatment monitoring by 18F-FDG PET/CT in patients with sarcomas: interobserver variability of quantitative parameters in treatment-induced changes in histopathologically responding and nonresponding tumors." Journal of nuclear medicine : official publication, Society of Nuclear Medicine **49**(7): 1038-1046.

Carson, R. E. (2005). Tracer Kinetic Modeling in PET. Positron Emission Tomography. D. L. Bailey, D. W. Townsend, P. E. Valk and M. N. Maisey, Springer London**:** 127-159.

Conrad, E. U., 3rd, H. D. Morgan, et al. (2004). "Fluorodeoxyglucose positron emission tomography scanning: basic principles and imaging of adult soft-tissue sarcomas." The Journal of bone and joint surgery. American volume **86-A Suppl 2**: 98-104.

Dimitrakopoulou-Strauss, A., L. G. Strauss, et al. (2010). "Impact of Dynamic (18)F-FDG PET on the Early Prediction of Therapy Outcome in Patients with High-Risk Soft-Tissue Sarcomas After Neoadjuvant Chemotherapy: A Feasibility Study." Journal of Nuclear Medicine **51**(4): 551-558.

Dimitrakopoulou-Strauss, A., L. G. Strauss, et al. (2001). "Dynamic PET F-18-FDG studies in patients with primary and recurrent soft-tissue sarcomas: Impact on diagnosis and correlation with grading." Journal of Nuclear Medicine **42**(5): 713-720.

DUO. (2011). "Digitale utgivelser ved UiO." Retrieved 01.09, 2011, from http://www.duo.uio.no/.

Eary, J. F., F. O'Sullivan, et al. (2008). "Spatial heterogeneity in sarcoma 18F-FDG uptake as a predictor of patient outcome." Journal of nuclear medicine : official publication, Society of Nuclear Medicine **49**(12): 1973-1979.

Fanning, D. (2011). "Coyote's Guide to IDL Programming." Retrieved 30.08, 2011, from http://www.idlcoyote.com/.

Gatenby, R. A. and R. J. Gillies (2004). "Why do cancers have high aerobic glycolysis?" Nature Reviews Cancer **4**(11): 891-899.

Jebsen, N. L., C. S. Trovik, et al. (2008). "Radiotherapy to improve local control regardless of surgical margin and malignancy grade in extremity and trunk wall soft tissue sarcoma: a Scandinavian sarcoma group study." International journal of radiation oncology, biology, physics **71**(4): 1196-1203.

Kamasak, M. E., C. A. Bouman, et al. (2005). "Direct reconstruction of kinetic parameter images from dynamic PET data." IEEE transactions on medical imaging **24**(5): 636-650.

Karam, I., S. Devic, et al. (2009). "PET/CT for radiotherapy treatment planning in patients with soft tissue sarcomas." International journal of radiation oncology, biology, physics **75**(3): 817-821.

Kasper, B., S. Dietrich, et al. (2008). "Early prediction of therapy outcome in patients with high-risk soft tissue sarcoma using positron emission tomography." Onkologie **31**(3): 107-112.

Keyes, J. W., Jr. (1995). "SUV: standard uptake or silly useless value?" Journal of nuclear medicine : official publication, Society of Nuclear Medicine **36**(10): 1836-1839.
Malinen, E., J. Rodal, et al. (2011). "Spatiotemporal analysis of tumor uptake patterns in dynamic (18)FDG-PET and dynamic contrast enhanced CT." Acta oncologica **50**(6): 873-882.
Matushansky, I. and R. G. Maki (2005). "Mechanisms of sarcomagenesis." Hematology/oncology clinics of North America **19**(3): 427-449, v.
Ohtake, T., N. Kosaka, et al. (1991). "Noninvasive method to obtain input function for measuring tissue glucose utilization of thoracic and abdominal organs." Journal of nuclear medicine : official publication, Society of Nuclear Medicine **32**(7): 1432-1438.
Phelps, M. E. (2004). PET : molecular imaging and its biological applications. New York, Springer.
Pugachev, A., S. Ruan, et al. (2005). "Dependence of FDG uptake on tumor microenvironment." International journal of radiation oncology, biology, physics **62**(2): 545-553.
Schuetze, S. M., B. P. Rubin, et al. (2005). "Use of positron emission tomography in localized extremity soft tissue sarcoma treated with neoadjuvant chemotherapy." Cancer **103**(2): 339-348.
Schwarzbach, M. H., U. Hinz, et al. (2005). "Prognostic significance of preoperative [18-F] fluorodeoxyglucose (FDG) positron emission tomography (PET) imaging in patients with resectable soft tissue sarcomas." Annals of surgery **241**(2): 286-294.
Shankar, L. K., J. M. Hoffman, et al. (2006). "Consensus recommendations for the use of 18F-FDG PET as an indicator of therapeutic response in patients in National Cancer Institute Trials." Journal of nuclear medicine : official publication, Society of Nuclear Medicine **47**(6): 1059-1066.
Siemens. (2009). "Inside Biograph TruePoint PET•CT." Retrieved 30.08, 2011, from http://www.medical.siemens.com/siemens/en_US/gg_nm_FBAs/files/broch/br_09_btruep_technology_insert.pdf.
Stahl, A., K. Ott, et al. (2004). "Comparison of different SUV-based methods for monitoring cytotoxic therapy with FDG PET." European journal of nuclear medicine and molecular imaging **31**(11): 1471-1478.
Tofts, P. S., G. Brix, et al. (1999). "Estimating kinetic parameters from dynamic contrast-enhanced T-1-weighted MRI of a diffusable tracer: Standardized quantities and symbols." Jmri-Journal of Magnetic Resonance Imaging **10**(3): 223-232.
Tortora, G. J. and S. R. Grabowski (2003). Principles of anatomy and physiology. New York, Wiley.
Weber, W. A. (2005). "PET for response assessment in oncology: radiotherapy and chemotherapy." British Journal of Radiology **78**: 42-49.
Wikipedia. (2011). Retrieved 30.08, 2011, from http://en.wikipedia.org/wiki/Positron_Emission_Tomography.
Zhao, S., Y. Kuge, et al. (2005). "Biologic correlates of intratumoral heterogeneity in 18F-FDG distribution with regional expression of glucose transporters and hexokinase-II in experimental tumor." Journal of nuclear medicine : official publication, Society of Nuclear Medicine **46**(4): 675-682.

Part III
Appendix

Appendix A: Anatomy

Anatomy (ana- = up, -tomy = process of cutting) is the science of body structures and their internal relationship. The medical science has developed its own language shared by all countries and based on latin and greek. The figures used are from a standard textbook on anatomy(Tortora and Grabowski 2003).

Relational **concepts**

The directional terms used to describe the human body come in pairs with opposite meanings. They are relative and only work when they describe one structures position relative another. That is the knee is superior to the ankle, though both are on the inferior part of the body.

Superior – inferior (Cranio-Caudal)	Towards top (head) – bottom
Anterior-Posterior (Ventral-Dorsal)	Towards front – back
Medial-Lateral	Towards – away from midline
Proximal – Distal (Limbs)	Close to– far from attachment on trunk
Superficial - Deep	Close to surface- deep inside

Skeleton

Many anatomical structures are defined according to their position relative to the skeleton, as the bones are rigid and rather fixed in their positions.

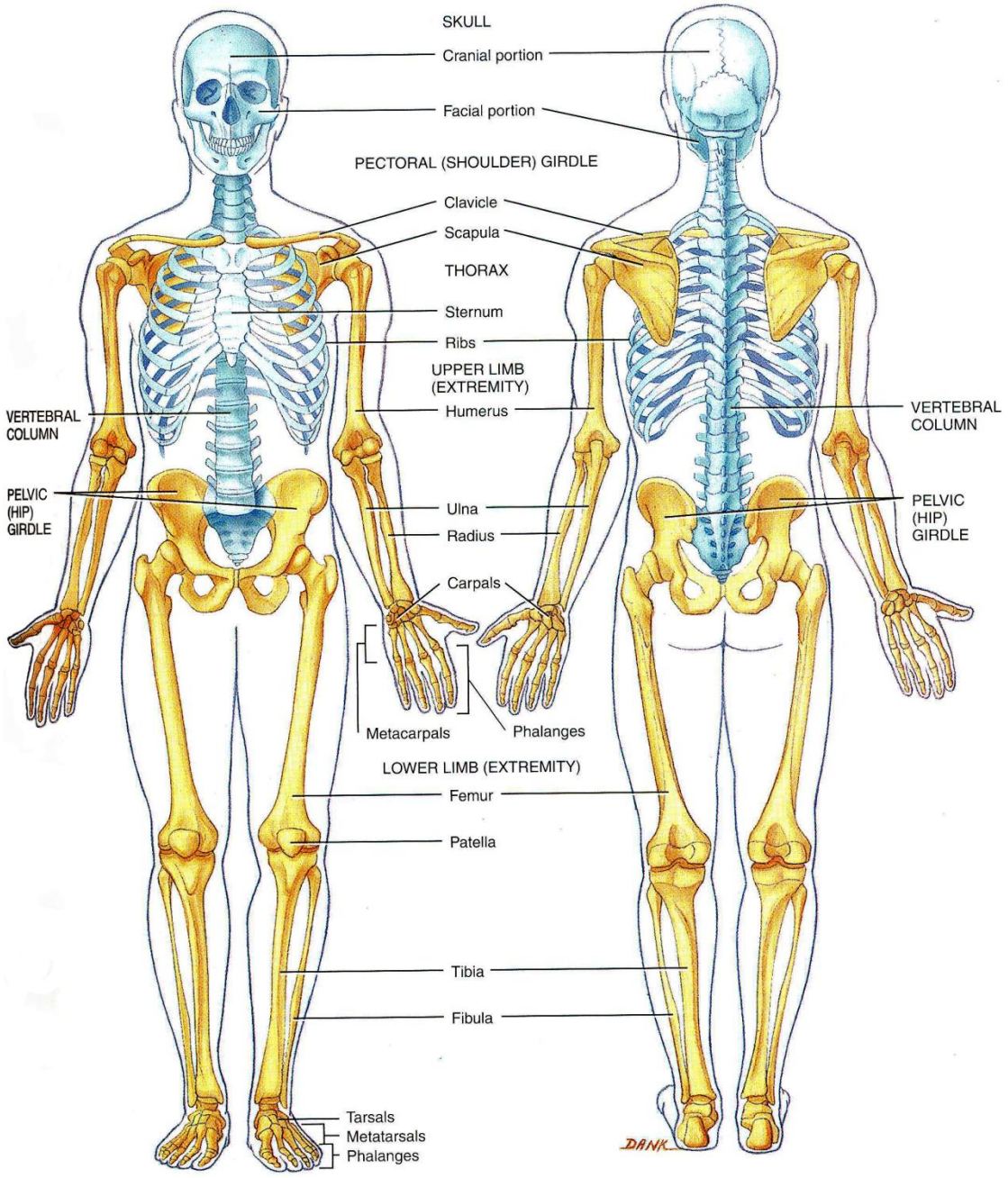

Figure 53. Anterior view of the skeleton to the left, and posterior view to the right.

Skeletal muscles

The skeletal muscles are attached to the skeleton and produce movement. In the external parts of the body they define the shape of the body, and are more fixed compared to fat tissue and as such describe the position of the tumor better even if it is situated in the fat itself.

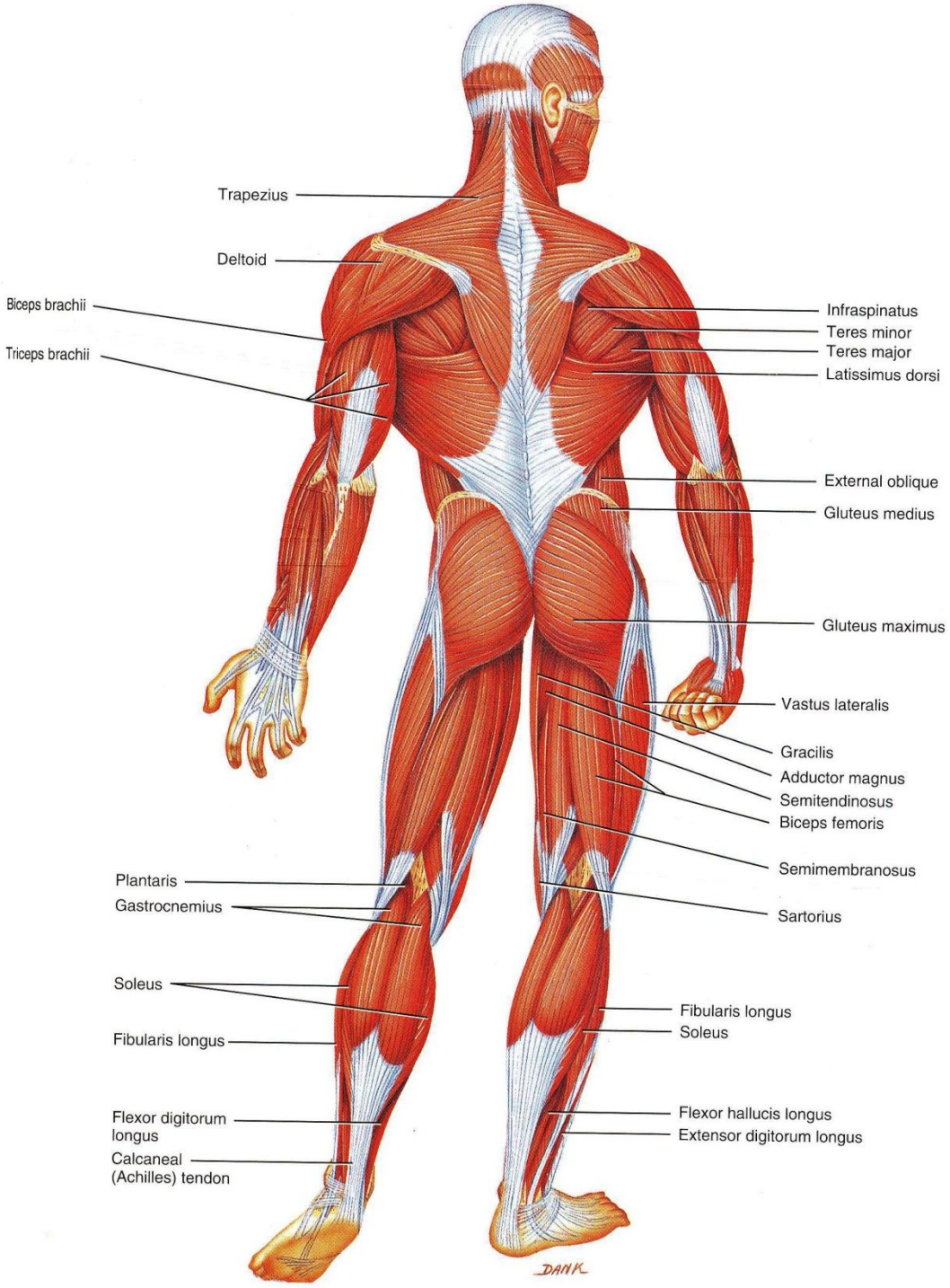

Figure 54. Posterior view of principal superficial skeletal muscles.

The circulatory system

The principal arteries of the human body.

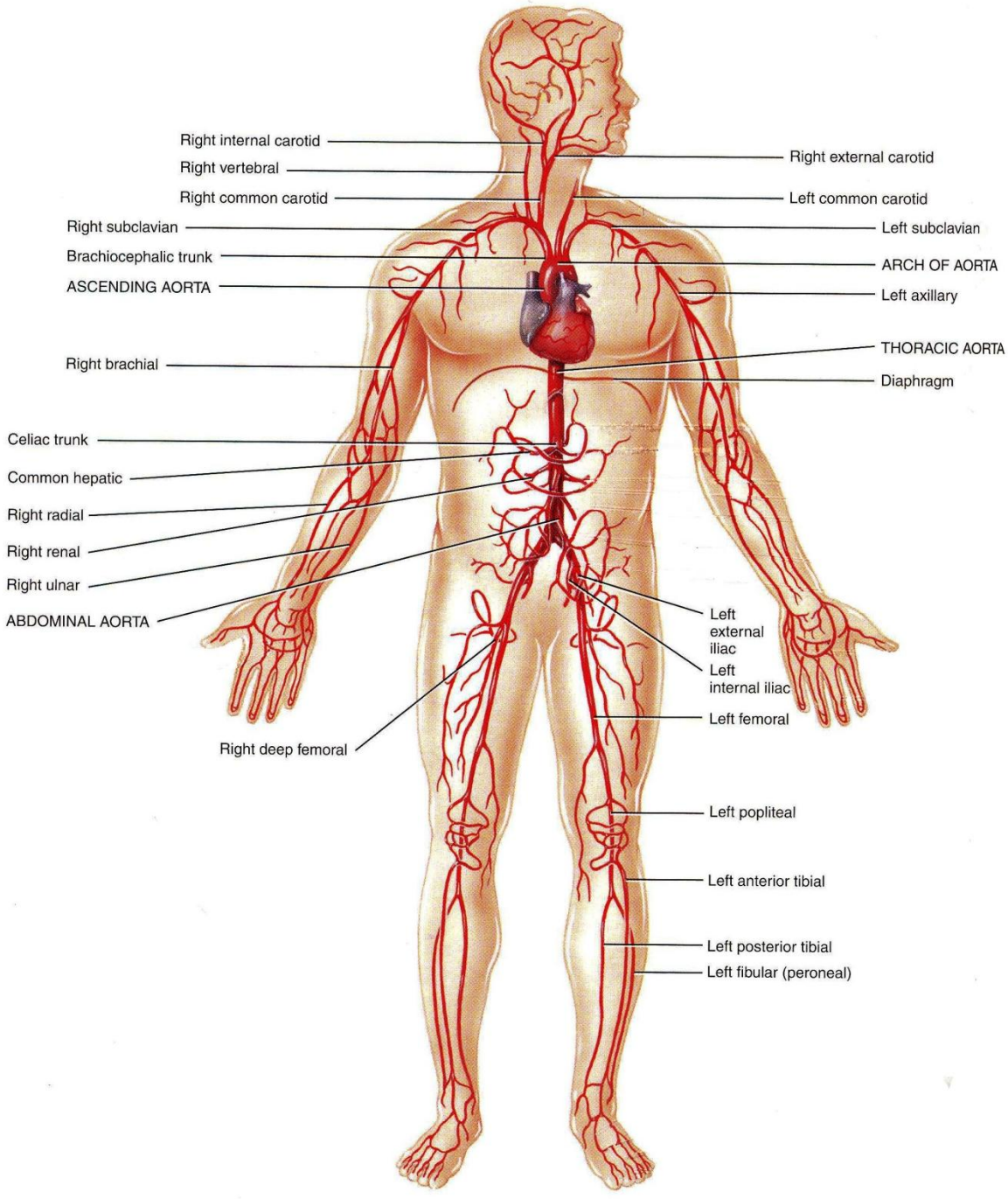

Figure 55. Anterior view of the principal branches of the aorta.

Appendix B: Mann-Whitney *U* statistic

The table that was used to calculate the significance level of the U values (Avery 2002).

n_1	n_2	0.2	0.1	0.05	0.02	0.01	0.002	0.001
6	2	11	12					
6	3	15	16	17				
6	4	19	21	22	23	24		
6	5	23	25	27	28	29		
6	6	27	29	31	33	34		
7	2	13	14					
7	3	17	19	20	21			
7	4	22	24	25	27	28		
7	5	27	29	30	32	34		
7	6	31	34	36	38	39	42	
7	7	36	38	41	43	45	48	49
8	2	14	15	16				
8	3	19	21	22	24			
8	4	25	27	28	30	31		
8	5	30	32	34	36	38	40	
8	6	35	38	40	42	44	47	48
8	7	40	43	46	49	50	54	55
8	8	45	49	51	55	57	60	62
9	1	9						
9	2	16	17	18				

Table 13 Critical values of the Mann-Whitney U statistic. To the left the values n_1 and n_2 represent the number of elements in class 1 and 2. To the right the level of significance and beneath it the U value required to reach the respective p value.

Appendix C: Scatter plots

Patient 1

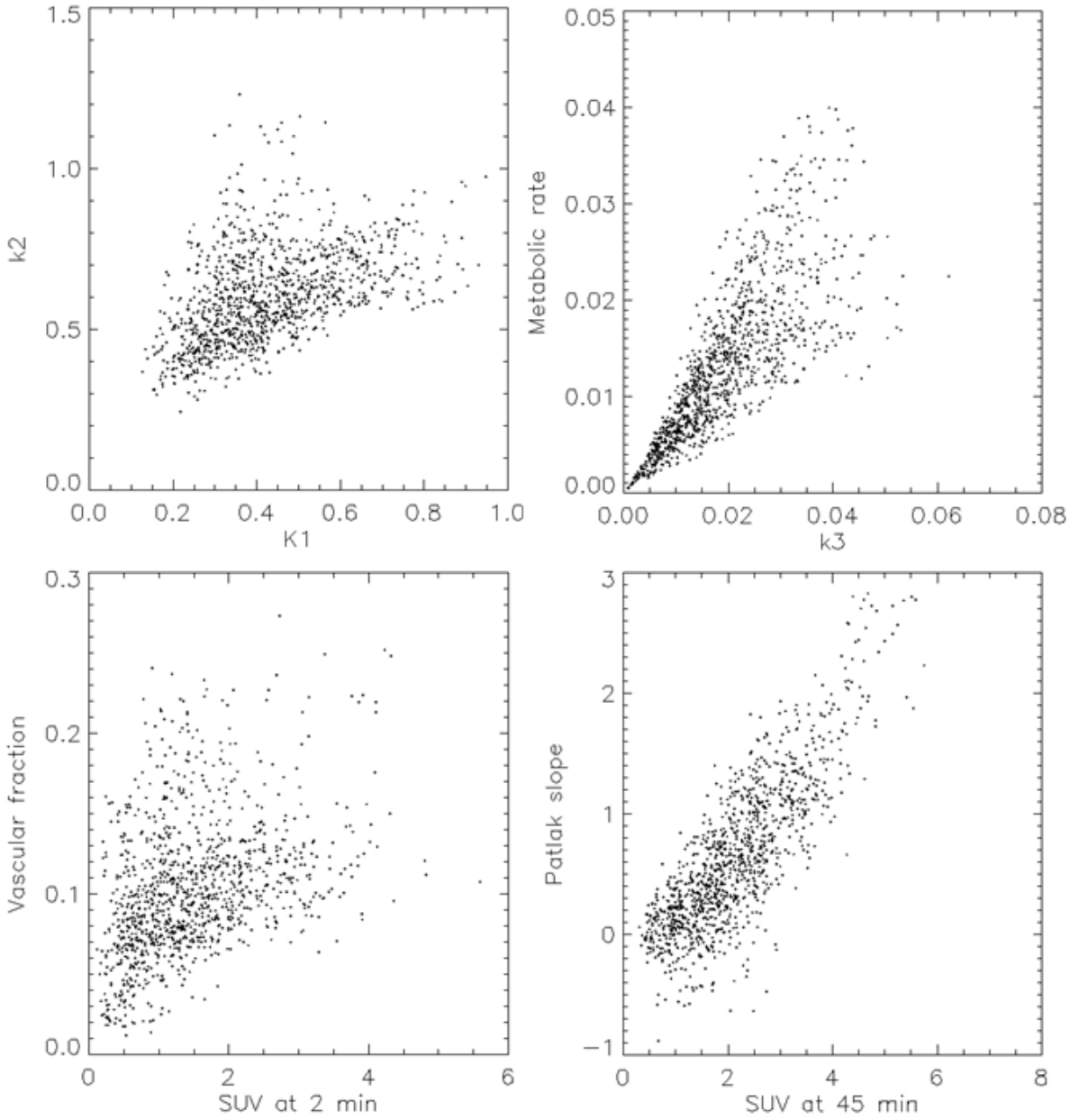

Figure 56. Scatter plots of the voxels of Patient 1.

Patient 2

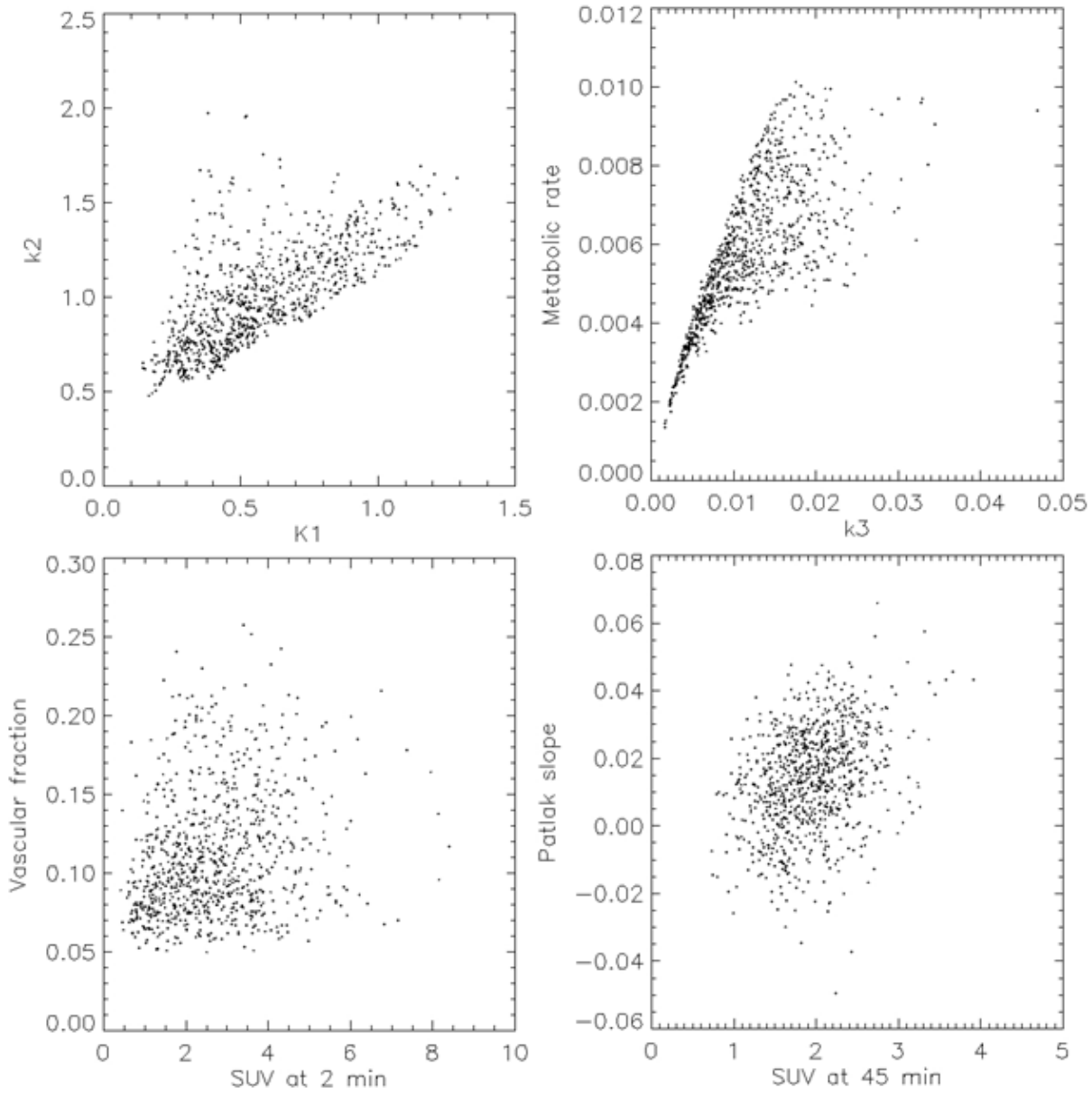

Figure 57. Scatter plots of the voxels of Patient 2.

Patient 3

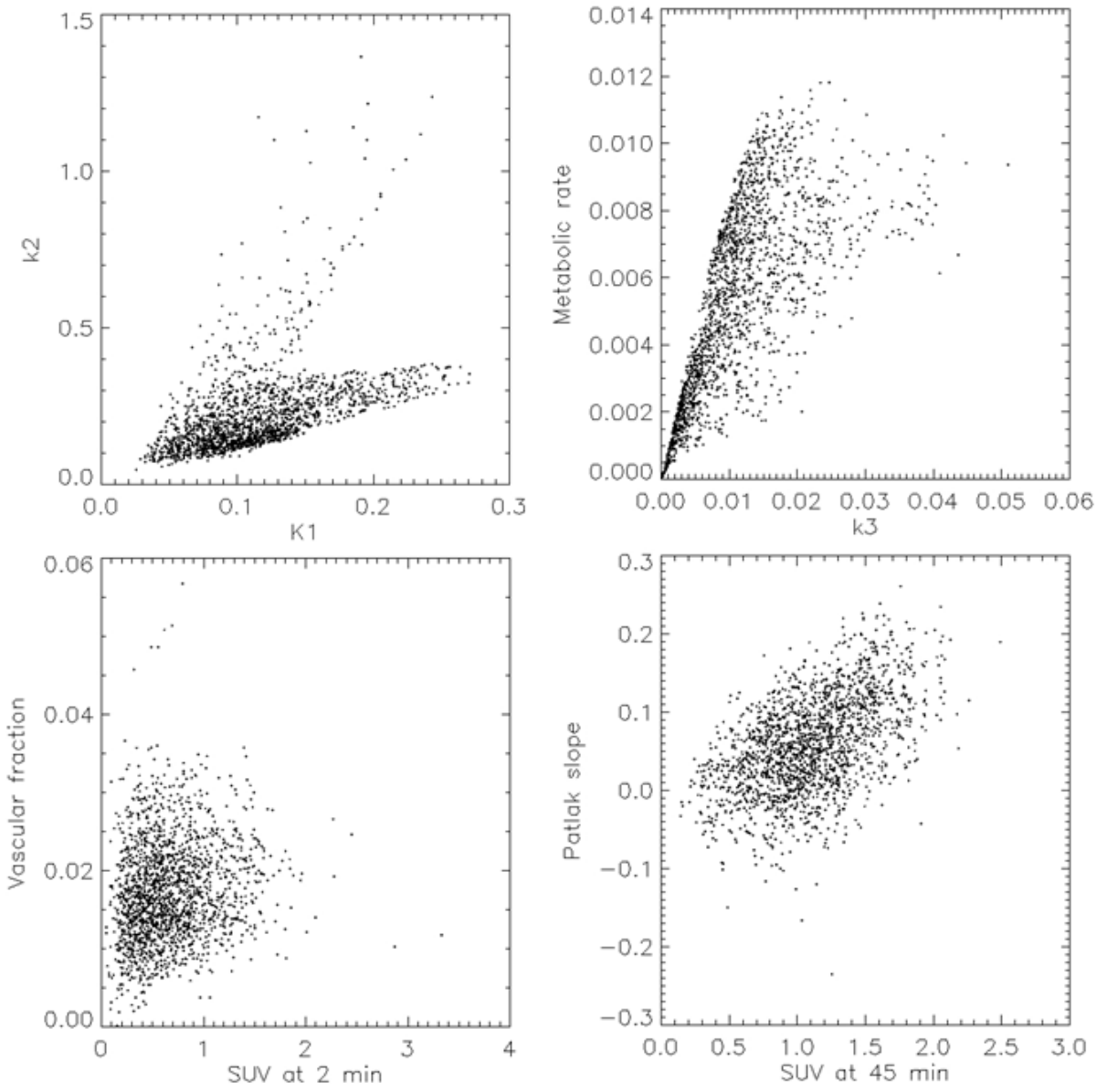

Figure 58. Scatter plots of the voxels of Patient 3.

Patient 4

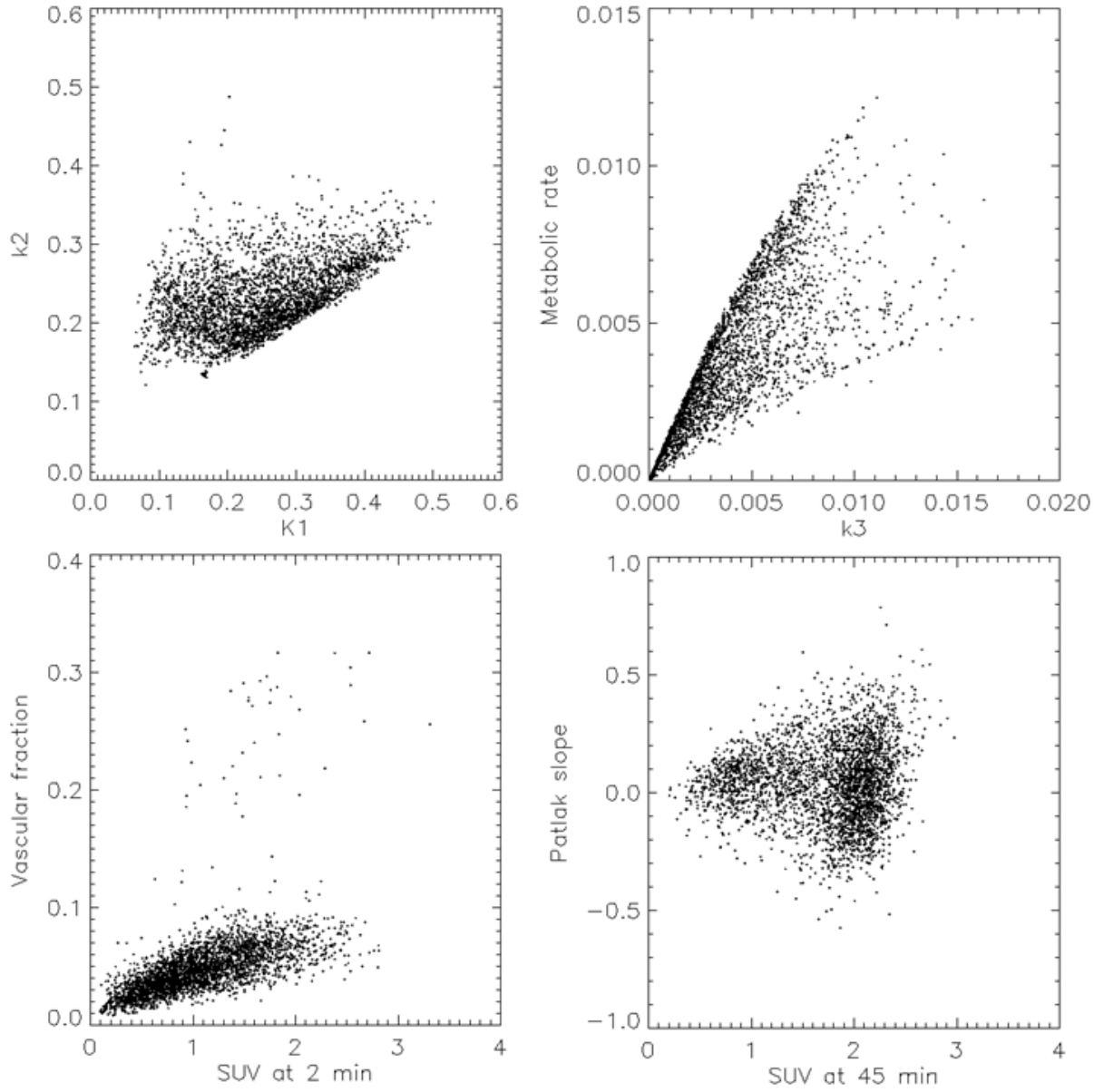

Figure 41. Scatter plots of the voxels of Patient 4.

Patient 5

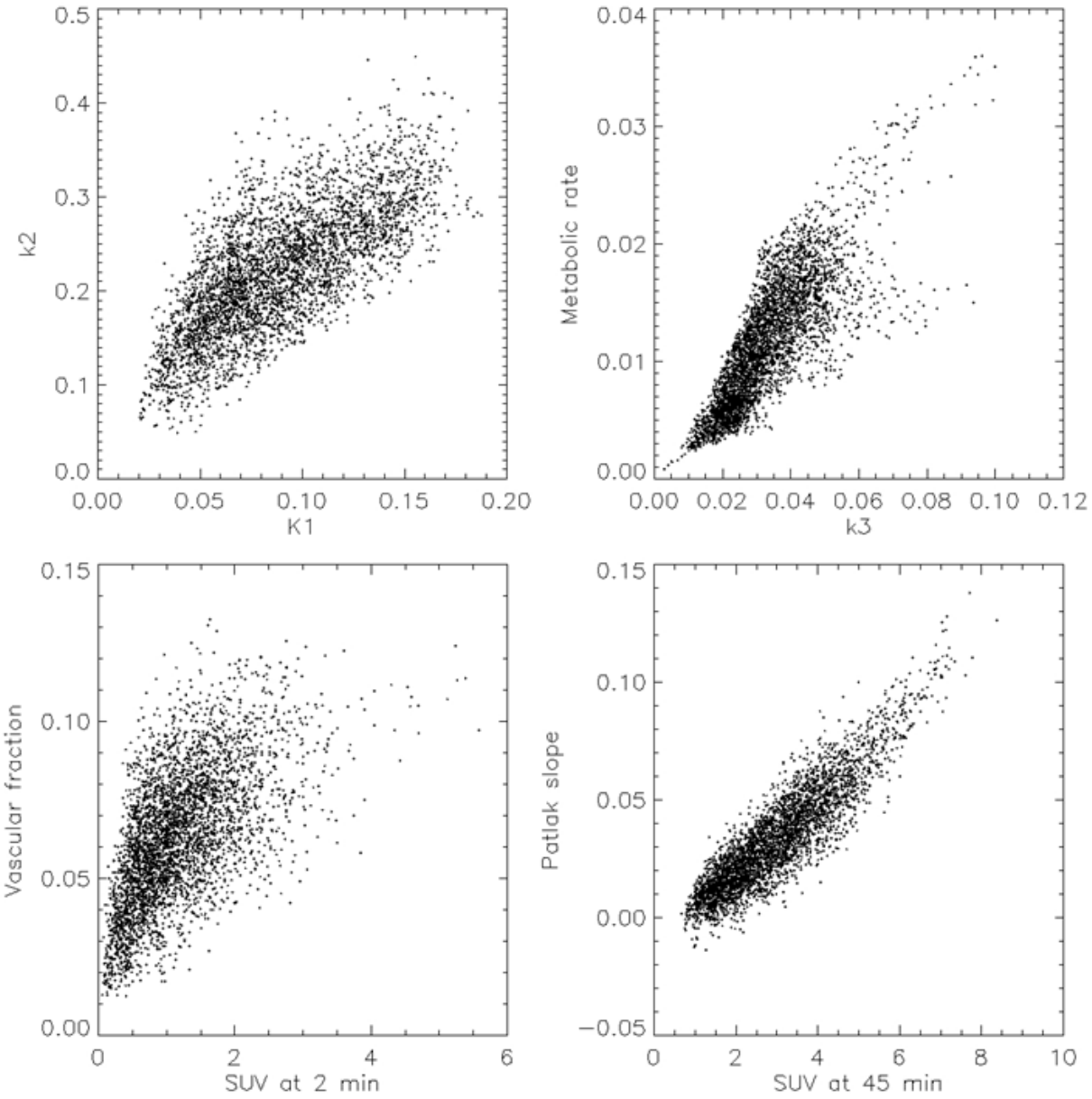

Figure 59. Scatter plots of the voxels of Patient 5.

Patient 6

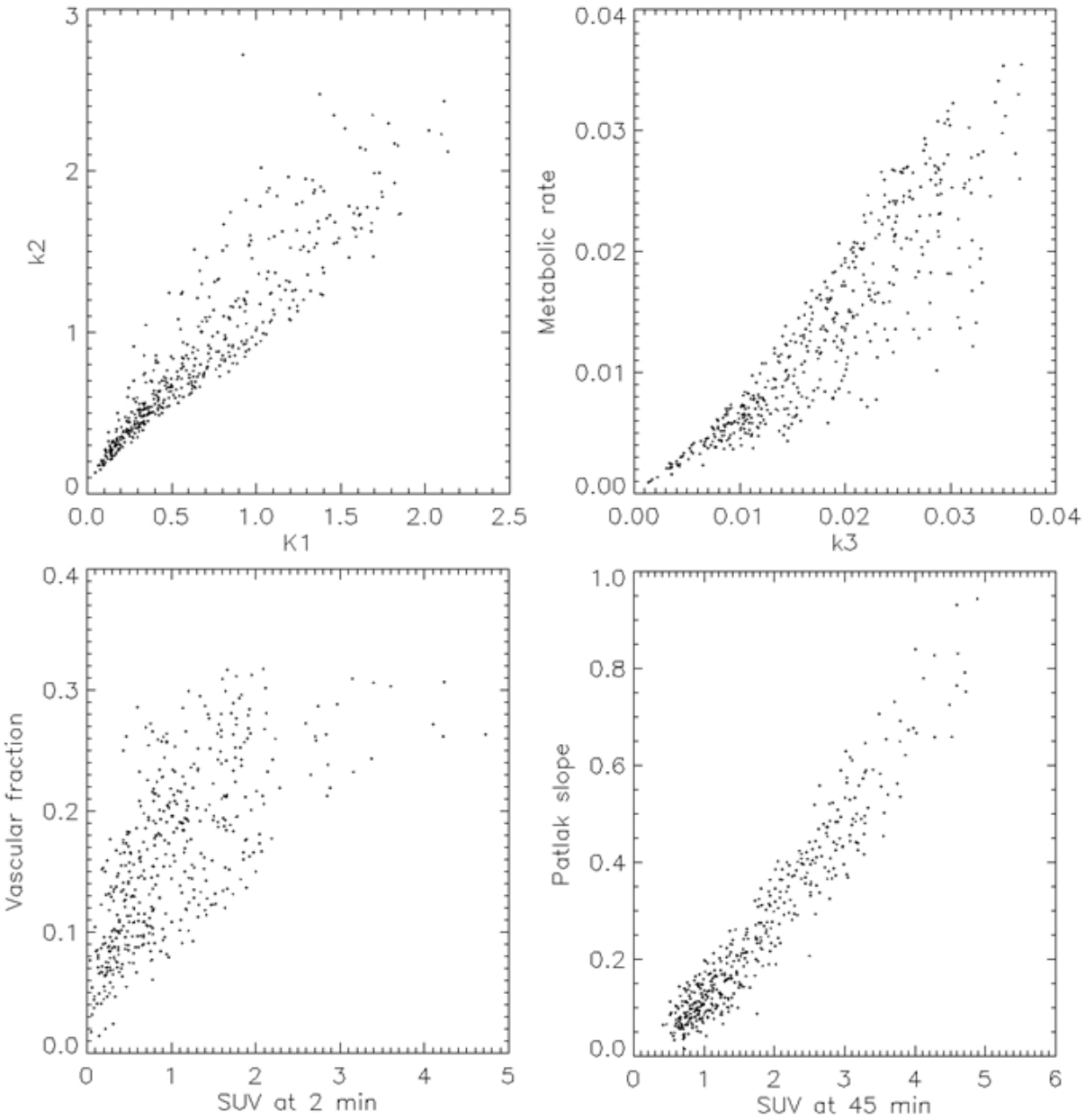

Figure 60. Scatter plots of the voxels of Patient 6.

Patient 7

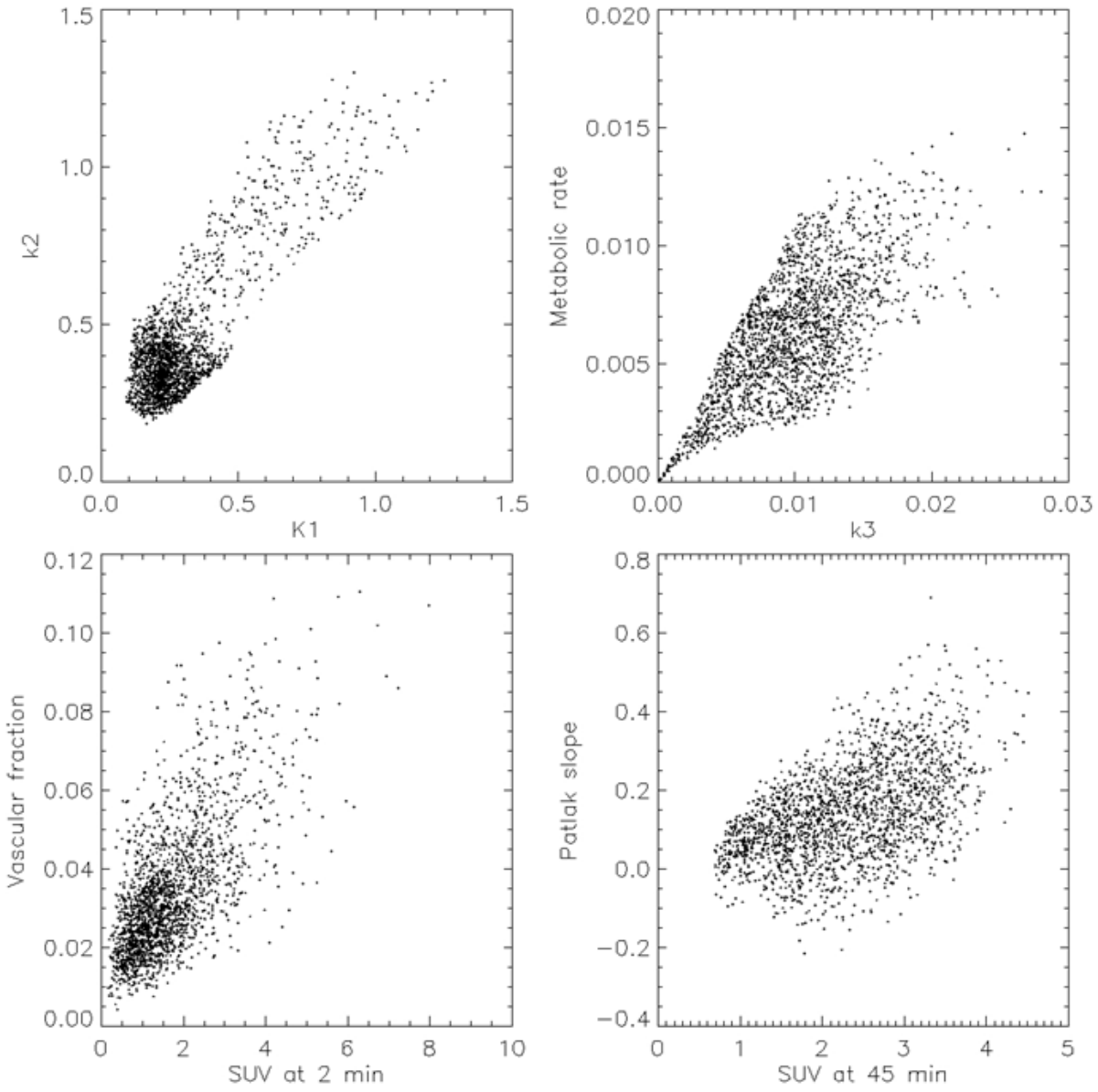

Figure 61. Scatter plots of the voxels of Patient 7.

Patient 8

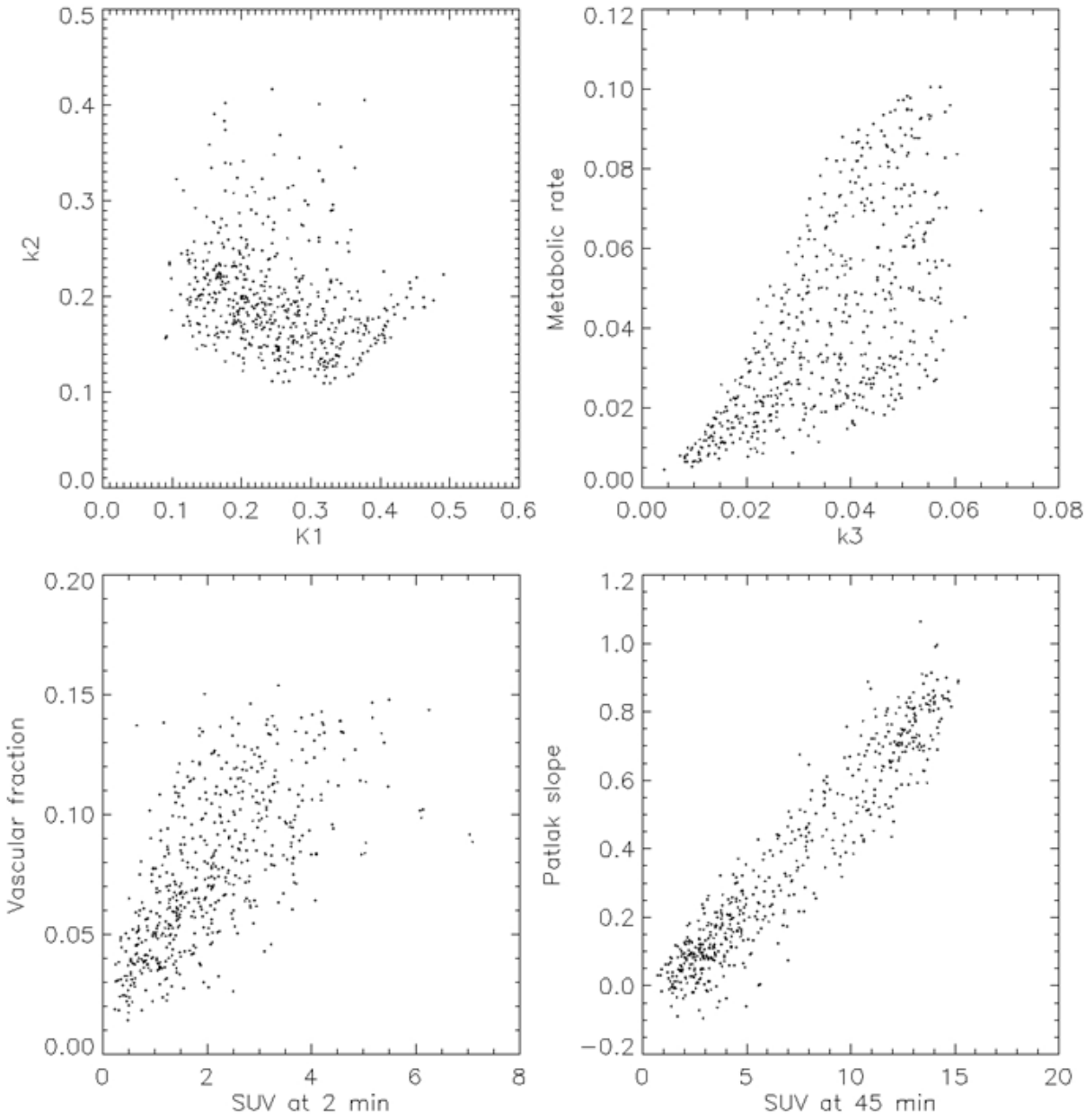

Figure 62. Scatter plots of the voxels of Patient 8.

Patient 9

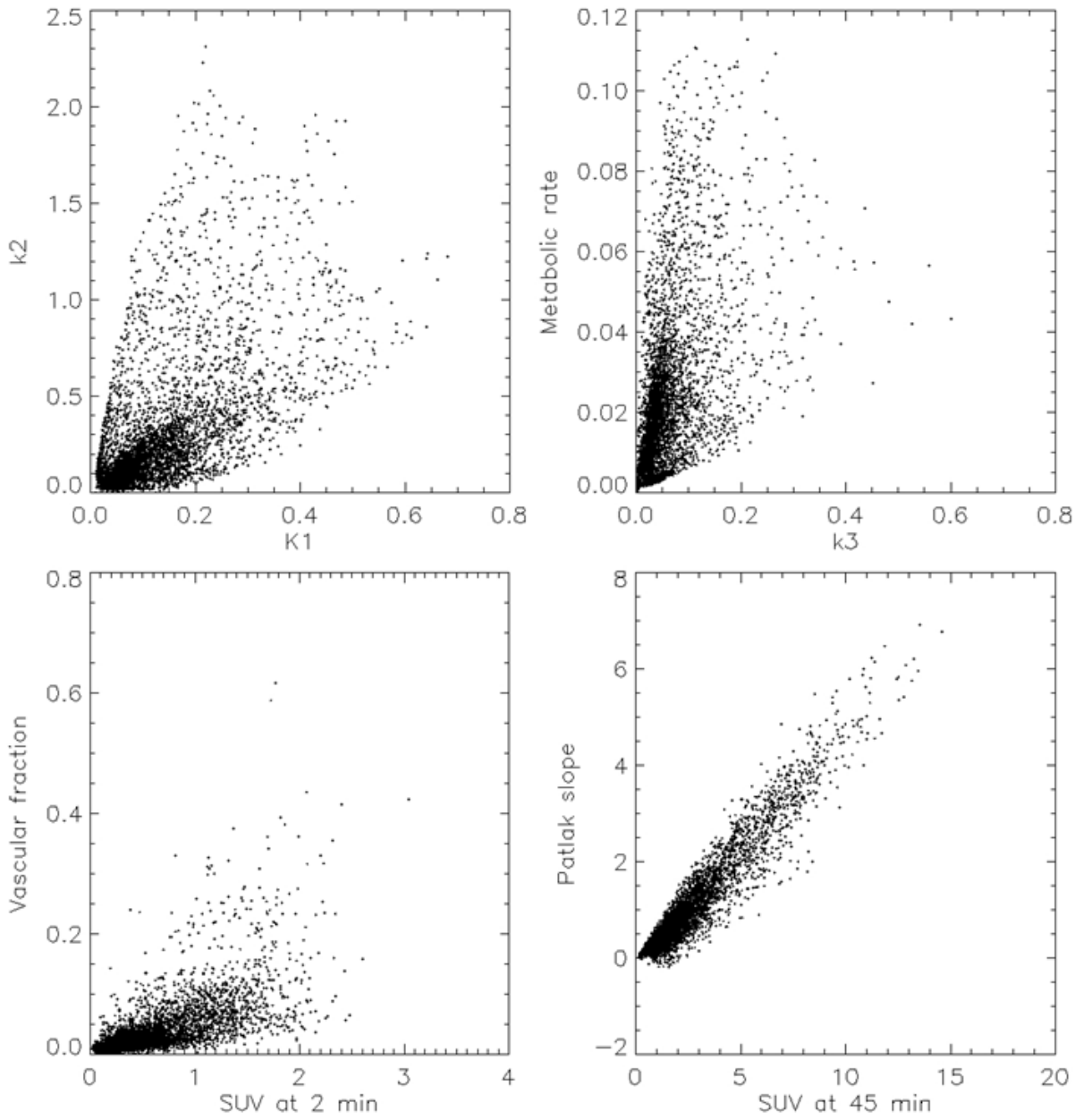

Figure 63. Scatter plots of the voxels of Patient 9.

Patient 10

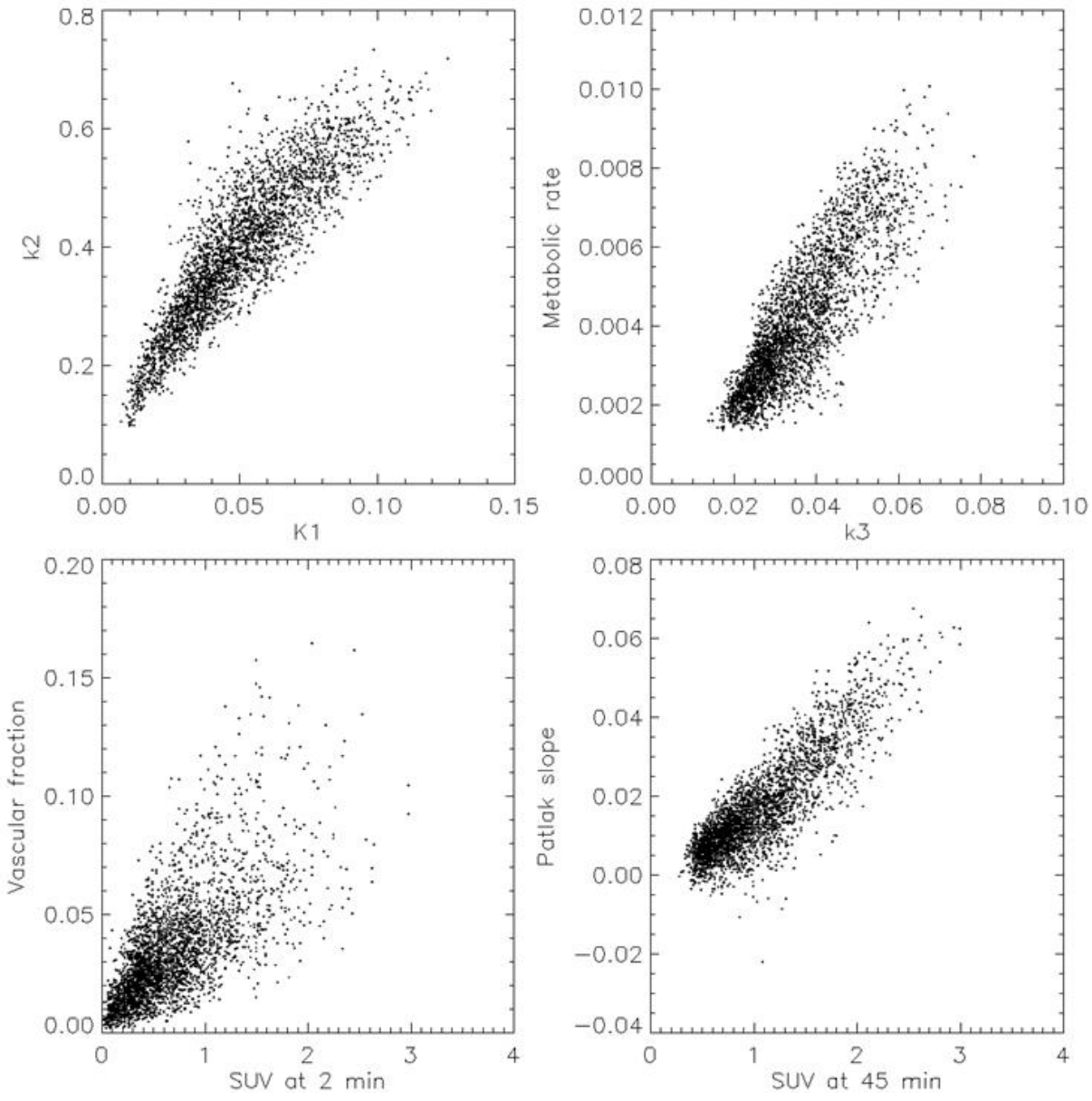

Figure 64. Scatter plots of the voxels of Patient 10.

Patient 11

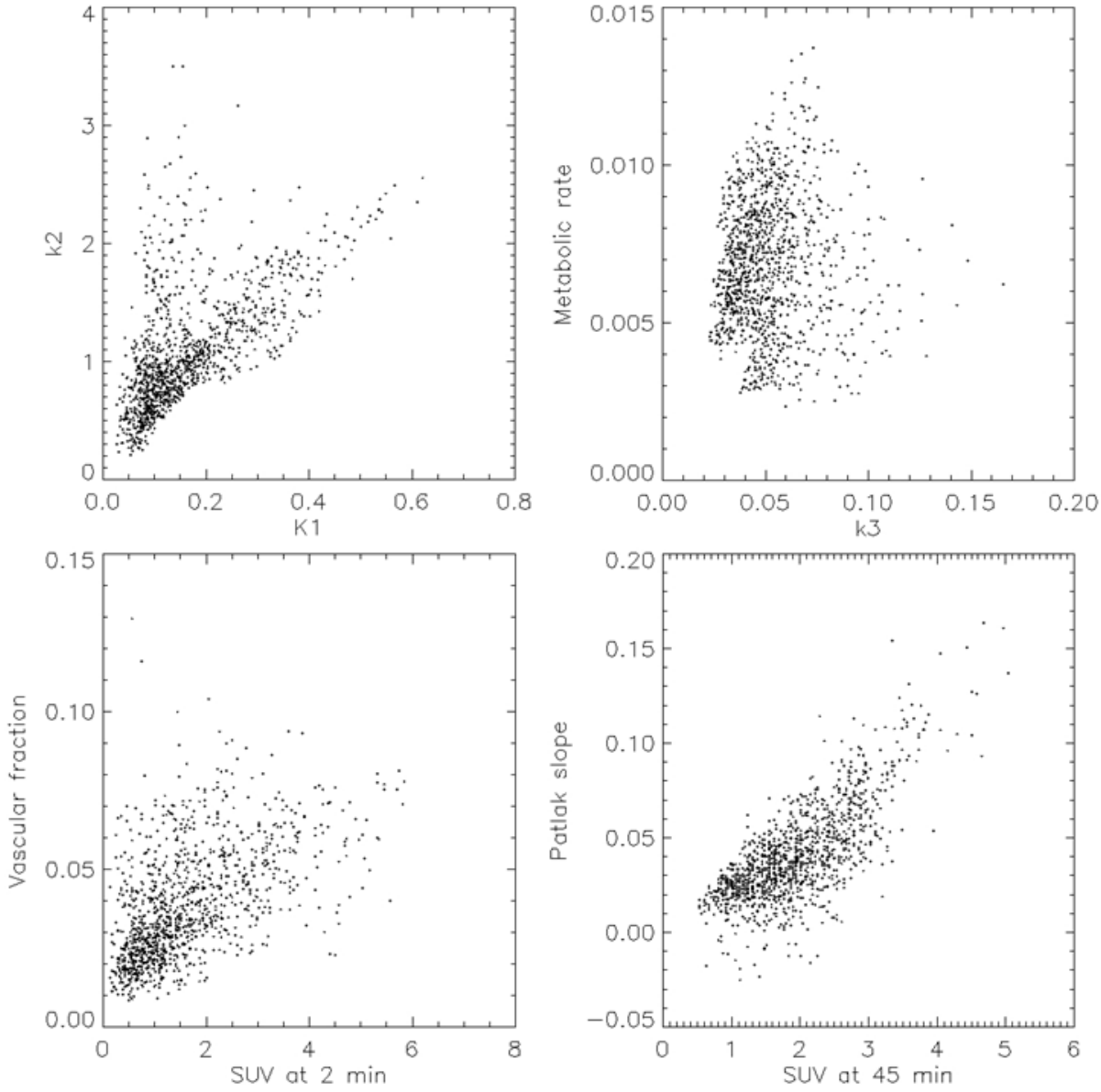

Figure 65. Scatter plots of the voxels of Patient 11.

Appendix D: Source code

Some truncations were done to conserve space, the full text can be found stored electronically at the University of Oslo (DUO 2011). To compile the software an external library (Coyote) has to be included (Fanning 2011).

GUI definition:

```
;The main procedure
PRO lgui

DEVICE, DECOMPOSED = 1

myBase = widget base(column=1, title = 'LVA', MBAR = fileBar, /CONTEXT_EVENTS)

file_menu      = widget button(fileBar, value='File', /menu)
file bttnOpen  = widget button(file_menu, value='Open PET', uvalue = 'open directory',
event pro='lgui loadDirectory')
file bttnCTOpen  = widget button(file_menu, value='Open CT', uvalue = 'open ct directory',
event_pro='lgui_loadCTDirectory')
file_bttnLoad  = widget button(file_menu, value='Load', uvalue = 'load file',
event_pro='lgui_load')
file bttnSave  = widget_button(file_menu, value='Save', uvalue = 'save file',
event pro='lgui save')
file bttnExport= widget button(file_menu, value='Export results', uvalue = 'Export',
event_pro='lgui_export')
file_bttnExit  = widget button(file_menu, value='Preferences', uvalue = 'Preferences',
event_pro='lgui_displayPreferences')
file bttnExit  = widget_button(file_menu, value='Exit', uvalue = 'Exit',
event_pro='lgui_exit')

tool_menu          = widget button(fileBar, value='Tools', /menu)
tool_bttnClearPlas = widget button(tool_menu, value='Clear plasma', uvalue = 'Clear plasma',
event_pro = 'lgui_resetPlasma')
tool bttnClearTumo = widget button(tool_menu, value='Clear tumor', uvalue = 'Clear tumor',
event pro = 'lgui resetTumor')
tool bttnCalculate = widget button(tool_menu, value = 'Calculate', uvalue='CalcCurve',
event_pro = 'lgui_calculateCurve')
tool_bttnCalculate = widget button(tool_menu, value = 'Calculate frame',
uvalue='CalcCurveFrame', event pro = 'lgui calculateCurve')
tool bttnCalculate = widget button(tool menu, value = 'Smooth calc',
uvalue='CalcCurveSmoothed', event pro = 'lgui calculateCurve')
tool_bttnCalculate = widget button(tool_menu, value = 'Smooth calc frame',
uvalue='CalcCurveSmoothedFrame', event_pro = 'lgui_calculateCurve')
tool_bttnCalculate = widget button(tool_menu, value = 'Calculate r2', uvalue='CalcPearsons',
event pro = 'lgui calculateCurve')
tool bttnCalculate = widget button(tool menu, value = 'Calculate r2 frame',
uvalue='CalcPearsonsFrame', event_pro = 'lgui_calculateCurve')
tool_bttnCalculate = widget button(tool_menu, value = 'Calculate Patlak frame',
uvalue='CalcPatlakFrame', event_pro = 'lgui_calculateCurve')
tool_bttnBreak     = widget button(tool_menu, value = 'Break', uvalue='LVABReak', event_pro =
'lgui break')
tool bttnLVABreak     = widget button(tool_menu, value = 'LVA Break', uvalue='LVAObjBReak',
event_pro = 'lgui_LVAbreak')

view menu             = widget button(fileBar, value='View', /menu )
view bttnTumor = widget button(view menu, value='Toggle Tumor ROI', uvalue = 'Toggle tumor
ROI', event pro = 'lgui displayTumorROI')
view bttnTumor = widget button(view menu, value='Toggle Tumor Edge', uvalue = 'Toggle tumor
ROI', event_pro = 'lgui_displayTumorEdgeROI')
view_bttnPlasma       = widget button(view_menu, value='Toggle Plasma ROI', uvalue = 'Toggle
plasma ROI', event pro = 'lgui displayPlasmaROI')
view bttnTissue = widget button(view menu, value='Toggle Tissue ROI', uvalue = 'Toggle tissue
ROI', event pro = 'lgui displayTissueROI')
view_bttnZoom  = widget button(view_menu, value='Toggle zoom', uvalue = 'Toggle zoom',
event_pro = 'lgui_toggleZoom')
view bttnPetsmo       = widget button(view menu, value='Toggle pet smoothing', uvalue =
'Toggle petsmut', event_pro = 'lgui_togglePETSmoothing')
```

```
view_bttnROIsmo       = widget_button(view_menu, value='Toggle tumor smoothing', uvalue =
'Toggle tumsmut', event_pro = 'lgui_toggleTumorROISmoothing')
view_bttnOutput       = widget_button(view_menu, value='Toggle tumor window', uvalue = 'Toggle
tumwin', event_pro = 'lgui_toggleTumorWindow')
view_bttnOutput       = widget_button(view_menu, value='Toggle output window', uvalue =
'Toggle outwin', event_pro = 'lgui_toggleOutputWindow')
;The output window
Window, XSIZE = 512, YSIZE = 512, /FREE, /PIXMAP
renderWindow = !D.Window
windowBase     = widget_base(mybase, /ROW, /align_center)
outputWindow   = widget_draw(windowBase, xsize = 512, ysize = 512, uvalue = 'outputWindow',
keyboard_events = 1, /WHEEL_EVENTS,  /BUTTON_EVENTS)
plotWindow     = widget_draw(windowBase, xsize = 512, ysize = 512, uvalue = 'plotWindow',
keyboard_events = 1, /BUTTON_EVENTS, /MOTION_EVENTS)

;First row
legendSize = 90

lwidth = 40
lheight = 15
;Scr_YSize= entryHeight,Scr_XSize=xsize
controlPanelBase    = widget_base(myBase, /ROW, /align_center)
firstLabelBase          = widget_base(controlPanelBase, /COLUMN, /align_center)
depthTitle          = widget_label(firstLabelBase, value = ' Depth: ' , uvalue= 'depth label',
xsize = lwidth, Scr_YSize = lheight)
timeLabel           = widget_label(firstLabelBase, value = ' Time: ' , uvalue= 'time label',
xsize = lwidth, Scr_YSize = lheight)
selectLabel           = widget_label(firstLabelBase, value = 'Select:' , uvalue= 'select
label', xsize = lwidth, Scr_YSize = lheight)

lwidth = 40
firstValueBase      = widget_base(controlPanelBase, /COLUMN, /align_center)
depthTitle          = widget_label(firstValueBase, value = string(0L, format = '(I3)'),
uvalue= 'depth label', xsize = lwidth, Scr_YSize = lheight)
timeLabel           = widget_label(firstValueBase, value = string(0L, format = '(I3)'),
uvalue= 'time label', xsize = lwidth, Scr_YSize = lheight)
dummyTitle          = widget_label(firstValueBase, value = ' ' , uvalue= 'dummy label', xsize
= lwidth, Scr_YSize = lheight)

lwidth = 120
firstSliderBase     = widget_base(controlPanelBase, /COLUMN, /align_center)
depthSelectSlider   = widget_slider(firstSliderBase, MAXIMUM = 1, MINIMUM = 0, uvalue='slide
depth', /suppress_value, xsize = lwidth, Scr_YSize = lheight)
timeSelectSlider    = widget_slider(firstSliderBase, MAXIMUM = 1, MINIMUM = 0, uvalue='slide
time', /suppress_value, xsize = lwidth, Scr_YSize = lheight)
selectorArray = make_array(10, /STRING)
selectorArray[0] = 'nothing'
selectorArray[1] = 'plasma voxel'
selectorArray[2] = 'tumor voxel'
selectorArray[3] = 'tissue voxel'
selectorArray[4] = 'current voxel'
selectorArray[5] = 'window center'
selectorArray[6] = 'tumor window'
selectorArray[7] = 'output window'
selectorArray[8] = 'Text position'
selectorArray[9] = 'Legend position'
selectionTypeDroplist = widget_droplist(firstSliderBase, value = selectorArray, uvalue='type
droplist')

lwidth = 45
labelBase           = widget_base(controlPanelBase, /COLUMN, /align_center)
PETNameLabel        = widget_label(labelBase, value = ' PET limits: ' , uvalue= 'time label',
xsize = lwidth, Scr_YSize = lheight)
MBNameLabel         = widget_label(labelBase, value = ' mb limits: ' , uvalue= 'time label',
xsize = lwidth, Scr_YSize = lheight)
VBNameLabel         = widget_label(labelBase, value = ' vb limits: ' , uvalue= 'time label',
xsize = lwidth, Scr_YSize = lheight)
CTNameLabel         = widget_label(labelBase, value = ' CT limits: ' , uvalue= 'time label',
xsize = lwidth, Scr_YSize = lheight)
psNameLabel         = widget_label(labelBase, value = ' ps limits: ' , uvalue= 'time label',
xsize = lwidth, Scr_YSize = lheight)
piNameLabel         = widget_label(labelBase, value = ' pi limits: ' , uvalue= 'time label',
xsize = lwidth, Scr_YSize = lheight)

lwidth = 60
valueBase           = widget_base(controlPanelBase, /COLUMN, /align_center)
```

```
PETLabel             = widget_label(valueBase, value = string(0L, format = '(I3)')+':'+
string(0L, format = '(I3)'), uvalue= 'time label', xsize = lwidth, Scr_YSize = lheight)
MBLabel             = widget_label(valueBase, value = string(0L, format = '(I3)')+':'+
string(0L, format = '(I3)'), uvalue= 'time label', xsize = lwidth, Scr_YSize = lheight)
VBLabel             = widget_label(valueBase, value = string(0L, format = '(I3)')+':'+
string(0L, format = '(I3)'), uvalue= 'time label', xsize = lwidth, Scr_YSize = lheight)
CTLabel             = widget_label(valueBase, value = string(0L, format = '(I3)')+':'+
string(0L, format = '(I3)'), uvalue= 'time label', xsize = lwidth, Scr_YSize = lheight)
psLabel             = widget_label(valueBase, value = string(0L, format = '(I3)')+':'+
string(0L, format = '(I3)'), uvalue= 'time label', xsize = lwidth, Scr_YSize = lheight)
piLabel             = widget_label(valueBase, value = string(0L, format = '(I3)')+':'+
string(0L, format = '(I3)'), uvalue= 'time label', xsize = lwidth, Scr_YSize = lheight)

lwidth = 120
sliderBase          = widget_base(controlPanelBase, /COLUMN, /align center)
minPetValSlider        = widget_slider(sliderBase, MAXIMUM = 149, MINIMUM = 0, value = 40,
uvalue='slide petmin val', /suppress_value, xsize = lwidth, Scr_YSize = lheight)
minMbValSlider        = widget_slider(sliderBase, MAXIMUM = 149, MINIMUM = 0, value = 40,
uvalue='slide mbmin val', /suppress_value, xsize = lwidth, Scr_YSize = lheight)
minVbValSlider        = widget_slider(sliderBase, MAXIMUM = 149, MINIMUM = 0, value = 40,
uvalue='slide vbmin val', /suppress value, xsize = lwidth, Scr YSize = lheight)
minCTValSlider        = widget slider(sliderBase, MAXIMUM = 1000, MINIMUM = 0, value = 850,
uvalue='slide ctmin val', /suppress_value, xsize = lwidth, Scr_YSize = lheight)
minpsValSlider        = widget_slider(sliderBase, MAXIMUM = 149, MINIMUM = 0, value = 40,
uvalue='slide psmin val', /suppress_value, xsize = lwidth, Scr_YSize = lheight)
minpiValSlider        = widget_slider(sliderBase, MAXIMUM = 149, MINIMUM = 0, value = 40,
uvalue='slide pimin val', /suppress_value, xsize = lwidth, Scr_YSize = lheight)

lwidth = 120
secondSliderBase    = widget_base(controlPanelBase, /COLUMN, /align_center)
maxPetValSlider        = widget_slider(secondSliderBase, MAXIMUM = 1200, MINIMUM = 150, value
= 500, uvalue='slide petmax val', /suppress value, xsize = lwidth, Scr YSize = lheight)
maxMbValSlider        = widget slider(secondSliderBase, MAXIMUM = 1200, MINIMUM = 150, value =
500, uvalue='slide mbmax val' , /suppress value, xsize = lwidth, Scr YSize = lheight)
maxVbValSlider        = widget_slider(secondSliderBase, MAXIMUM = 1000, MINIMUM = 150, value =
500, uvalue='slide vbmax val' , /suppress_value, xsize = lwidth, Scr_YSize = lheight)
maxCTValSlider        = widget_slider(secondSliderBase, MAXIMUM = 2000, MINIMUM = 1000, value
= 1300, uvalue='slide ctmax val', /suppress value, xsize = lwidth, Scr YSize = lheight)
maxpsValSlider        = widget slider(secondSliderBase, MAXIMUM = 1200, MINIMUM = 150, value =
500, uvalue='slide psmax val', /suppress value, xsize = lwidth, Scr YSize = lheight)
maxpiValSlider        = widget_slider(secondSliderBase, MAXIMUM = 1200, MINIMUM = 150, value =
500, uvalue='slide pimax val', /suppress_value, xsize = lwidth, Scr_YSize = lheight)

lwidth = 45
thirdLabelBase      = widget_base(controlPanelBase, /COLUMN, /align_center)
K1NameLabel         = widget_label(thirdLabelBase, value = ' K1 limits: ' , uvalue= 'time
label', xsize = lwidth, Scr_YSize = lheight)
k2NameLabel         = widget_label(thirdLabelBase, value = ' k2 limits: ' , uvalue= 'time
label', xsize = lwidth, Scr YSize = lheight)
k3NameLabel         = widget label(thirdLabelBase, value = ' k3 limits: ' , uvalue= 'time
label', xsize = lwidth, Scr_YSize = lheight)

lwidth = 60
thirdValueBase      = widget_base(controlPanelBase, /COLUMN, /align_center)
K1Label             = widget_label(thirdValueBase, value = string(0L, format = '(I3)')+':'+
string(0L, format = '(I3)'), uvalue= 'time label', xsize = lwidth, Scr_YSize = lheight)
k2Label             = widget_label(thirdValueBase, value = string(0L, format = '(I3)')+':'+
string(0L, format = '(I3)'), uvalue= 'time label', xsize = lwidth, Scr_YSize = lheight)
k3Label             = widget_label(thirdValueBase, value = string(0L, format = '(I3)')+':'+
string(0L, format = '(I3)'), uvalue= 'time label', xsize = lwidth, Scr_YSize = lheight)

lwidth = 120
thirdSliderBase     = widget_base(controlPanelBase, /COLUMN, /align_center)
minK1ValSlider        = widget_slider(thirdSliderBase, MAXIMUM = 149, MINIMUM = 0, value = 40,
uvalue='slide K1min val', /suppress value, xsize = lwidth, Scr YSize = lheight)
mink2ValSlider        = widget slider(thirdSliderBase, MAXIMUM = 149, MINIMUM = 0, value = 40,
uvalue='slide k2min val', /suppress_value, xsize = lwidth, Scr_YSize = lheight)
mink3ValSlider        = widget_slider(thirdSliderBase, MAXIMUM = 149, MINIMUM = 0, value = 40,
uvalue='slide k3min val', /suppress_value, xsize = lwidth, Scr_YSize = lheight)

thirdSecSliderBase  = widget_base(controlPanelBase, /COLUMN, /align center)
maxK1ValSlider        = widget slider(thirdSecSliderBase, MAXIMUM = 1200, MINIMUM = 150, value
= 500, uvalue='slide K1max val', /suppress_value, xsize = lwidth, Scr_YSize = lheight)
maxk2ValSlider        = widget_slider(thirdSecSliderBase, MAXIMUM = 1200, MINIMUM = 150, value
= 500, uvalue='slide k2max val', /suppress_value, xsize = lwidth, Scr_YSize = lheight)
maxk3ValSlider        = widget_slider(thirdSecSliderBase, MAXIMUM = 1200, MINIMUM = 150, value
```

```
= 500, uvalue='slide k3max val', /suppress_value, xsize = lwidth, Scr_YSize = lheight)

;Second row
buttonBase = widget_base(myBase, uvalue='Display plasma', /ROW, /align_center)
buttonSize = 65
;optionsCheckers = CW_BGROUP(buttonBase, ['red'], /ROW, /NONEXCLUSIVE, /RETURN_NAME)

;toggleButton = widget_button(buttonBase, value = 'Toggle', uvalue='TogglePlasma', event_pro =
'lgui togglePlasma', xsize = buttonSize)
;resetButton = widget button(buttonBase, value = 'Reset', uvalue='Reset', event_pro =
'lgui_resetPlasma', xsize = buttonSize)

;Output window context base
contextBase      = widget base(myBase, /context_menu)
dispFramButton   = widget button(contextBase, value='PET Frame',       uvalue='Display frame',
event pro='lgui contextBarEvent')
dispCTFramButton = widget button(contextBase, value='CT Frame',        uvalue='Display CT
frame',       event_pro='lgui_contextBarEvent')
dispBlenFraBttn  = widget button(contextBase, value='SUV/CT composite',uvalue='Display blended
frame',  event pro='lgui contextBarEvent')
dispBlenFraBttn  = widget button(contextBase, value='k1/CT composite',uvalue='Display k1
blended frame',  event pro='lgui contextBarEvent')
dispBlenFraBttn  = widget button(contextBase, value='k2/CT composite',uvalue='Display k2
blended frame',  event_pro='lgui_contextBarEvent')
dispBlenFraBttn  = widget button(contextBase, value='k3/CT composite',uvalue='Display k3
blended frame',  event pro='lgui contextBarEvent')
dispBlenFraBttn  = widget button(contextBase, value='vb/CT composite',uvalue='Display vb
blended frame',  event pro='lgui contextBarEvent')
dispBlenFraBttn  = widget button(contextBase, value='Metabolic/CT composite',uvalue='Display
mb blended frame',  event_pro='lgui_contextBarEvent')
dispBlenFraBttn  = widget button(contextBase, value='patlak slope/CT
composite',uvalue='Display pSlope blended frame',  event pro='lgui contextBarEvent')
dispBlenFraBttn  = widget button(contextBase, value='patlak intercept/CT
composite',uvalue='Display pInter blended frame',  event pro='lgui contextBarEvent')
histogramButton  = widget button(contextBase, value='Histogram',       uvalue='Display
histogram',      event_pro='lgui_contextBarEvent')
plasmaButton     = widget button(contextBase, value='Plasma average',  uvalue='Display
plasma',         event pro='lgui contextBarEvent')
tumorButton      = widget button(contextBase, value='Tumor average',   uvalue='Display tumor
plot',     event pro='lgui contextBarEvent')
tissueButton     = widget button(contextBase, value='Tissue average',  uvalue='Display tissue
plot',    event_pro='lgui_contextBarEvent')
patlakButton     = widget button(contextBase, value='Patlak',          uvalue='Display patlak'
,       event pro='lgui contextBarEvent')
k1Button         = widget button(contextBase, value='K1 frame',        uvalue='Display k1
frame',       event_pro='lgui_contextBarEvent')
k2Button         = widget button(contextBase, value='K2 frame',        uvalue='Display k2
frame',       event_pro='lgui_contextBarEvent')
k3Button         = widget button(contextBase, value='K3 frame',        uvalue='Display k3
frame',       event pro='lgui contextBarEvent')
;k4Button        = widget_button(contextBase, value='K4 frame',        uvalue='Display k4
frame',       event_pro='lgui_contextBarEvent')
vbButton         = widget button(contextBase, value='VB frame',        uvalue='Display vb
frame',       event_pro='lgui_contextBarEvent')
vbButton         = widget button(contextBase, value='R2 frame',        uvalue='Display r2
frame',       event pro='lgui contextBarEvent')
pixelButton      = widget button(contextBase, value='Voxel function',  uvalue='Display voxel
function', event_pro='lgui_contextBarEvent')
defroiButton     = widget button(contextBase, value='Define ROI',      uvalue='Define region',
event_pro='lgui_contextBarEvent')

secondaryContextBase    = widget base(myBase, /context menu)
dispFramButton          = widget button(secondaryContextBase, value='PET Frame',
uvalue='Display sec frame',     event_pro='lgui_contextBarEvent')
dispCTFramButton        = widget button(secondaryContextBase, value='CT Frame',
uvalue='Display sec CT frame',  event pro='lgui contextBarEvent')
dispBlenFraBttn         = widget button(secondaryContextBase, value='SUV/CT
composite',uvalue='Display sec blended frame',  event_pro='lgui_contextBarEvent')
dispBlenFraBttn         = widget button(secondaryContextBase, value='k1/CT
composite',uvalue='Display sec k1 blended frame',  event pro='lgui contextBarEvent')
dispBlenFraBttn         = widget button(secondaryContextBase, value='k2/CT
composite',uvalue='Display sec k2 blended frame',  event pro='lgui contextBarEvent')
dispBlenFraBttn         = widget button(secondaryContextBase, value='k3/CT
composite',uvalue='Display sec k3 blended frame',  event_pro='lgui_contextBarEvent')
dispBlenFraBttn         = widget button(secondaryContextBase, value='vb/CT
composite',uvalue='Display sec vb blended frame',  event_pro='lgui_contextBarEvent')
```

```
dispBlenFraBttn         = widget button(secondaryContextBase, value='Metabolic/CT
composite',uvalue='Display sec mb blended frame',  event_pro='lgui_contextBarEvent')
disppSlopeButton        = widget button(secondaryContextBase, value='patlak slope/CT
composite', uvalue='Display sec pSlope blend', event pro='lgui contextBarEvent')
disppInterButton        = widget button(secondaryContextBase, value='patlak intercept/CT
composite', uvalue='Display sec pInter blend', event pro='lgui contextBarEvent')
plasmaButton            = widget button(secondaryContextBase, value='Plasma average',
uvalue='Display sec plasma',    event_pro='lgui_contextBarEvent')
k1Button                = widget button(secondaryContextBase, value='k1',   uvalue='Display
sec k1',    event pro='lgui contextBarEvent')
k2Button                = widget button(secondaryContextBase, value='k2',   uvalue='Display
sec k2',    event_pro='lgui_contextBarEvent')
k3Button                = widget button(secondaryContextBase, value='k3',   uvalue='Display
sec k3',    event_pro='lgui_contextBarEvent')
r2Button                = widget button(secondaryContextBase, value='r2',   uvalue='Display
sec r2',    event pro='lgui_contextBarEvent')
;Activate the hierarchy
widget_control, myBase, /realize

widget control, outputWindow, get value=outputWinid
widget_control, plotWindow, get_value=plotWinid

; Our data struct
lguiData = {$
depthLabel:depthTitle,$
timeLabel:timeLabel, $
PETLabel:PETLabel,$
MBLabel:MBLabel,$
VBLabel:VBLabel,$
K1Label:K1Label,$
k2Label:k2Label,$
k3Label:k3Label,$
CTLabel:CTLabel,$
psLabel:psLabel,$
piLabel:piLabel,$
minPetSlider:minPetValSlider,$
minMbSlider:minMbValSlider,$
minVbSlider:minVbValSlider,$
maxPetSlider:maxPetValSlider,$
maxMbSlider:maxMbValSlider,$
maxVbSlider:maxVbValSlider,$
minK1Slider:minK1ValSlider,$
mink2Slider:mink2ValSlider,$
mink3Slider:mink3ValSlider,$
minCTSlider:minCTValSlider,$
minpsSlider:minpsValSlider,$
minpiSlider:minpiValSlider,$
maxK1Slider:maxK1ValSlider,$
maxk2Slider:maxk2ValSlider,$
maxk3Slider:maxk3ValSlider,$
maxCTSlider:maxCTValSlider,$
maxpsSlider:maxpsValSlider,$
maxpiSlider:maxpiValSlider,$
outputWinid:outputWinid,$
plotWinid:plotWinid,$
renderWinid:renderWindow,$
activeWindow:'outputWindow',$
selectType:0L,$
displayPlasma:1L,$
tumorROISmoothing:0L,$
tumorEdge:1B,$
sliceDataPtr:OBJ NEW('LVA SliceData'),$
timeSlider:timeSelectSlider,$
depthSlider:depthSelectSlider,$
contextMenuBase:contextBase,$
secondaryContextBase:secondaryContextBase,$
primaryWinDisplayMode:11L,$
secondWinDisplayMode:21L,$
rectangleXpoint:-1,$
rectangleYPoint:-1,$
rectangleFirstPoint:1$
}

;Our actual storage
dataPtr = ptr new(lguiData)
(*dataPtr).sliceDataPtr->LVA_Constructor
```

```
widget_control, myBase, set_uvalue=dataPtr

; Start the manager
xmanager, 'lgui', mybase, cleanup = 'lgui_cleanup', /no_block

COMMON T, draw
draw = plotWindow

END
```

GUI event processing and helper functions:

```
PRO lgui_event, EVENT
if(TAG_NAMES(event, /STRUCTURE_NAME) EQ 'WIDGET_CONTEXT') then begin
print, 'context trap'
return
endif
widget control, event.top, get uvalue=dataPtr
widget_control, event.id, get_uvalue=widget

if(ptr_valid(dataPtr) eq 0) then print, 'Corrupt data pointer'

case widget of
'outputWindow' : begin
(*dataPtr).activeWindow = 'outputWindow'

if(event.type eq 5 AND event.press gt 0) then begin
        ;print, event.ch
        ;Ctrl keystrokes
        if(event.ch eq 3) then lgui_calculateCurve, event          ;c
        if(event.ch eq 12) then lgui_load, event                   ;l
        if(event.ch eq 15) then lgui_loadDirectory, event          ;o
        if(event.ch eq 16) then lgui_displayPreferences, event     ;p
        if(event.ch eq 17) then lgui_exit, event                   ;q
        if(event.ch eq 19) then lgui_save, event                   ;s
        if(event.ch eq 20) then begin                             ;t
                (*dataPtr).SliceDataPtr->LVA_ToggleSUVTimeAveraging
                lgui_displayFrame, dataPtr
        endif

        ;Normal keystrokes
        if(event.ch eq 103) then begin                                          ;g
                (*dataPtr).SliceDataPtr->LVA_DecreaseTime
                lgui displayFrame, dataPtr
                lgui_updateLabels, dataPtr
        endif
        if(event.ch eq 106) then begin                                          ;j
                (*dataPtr).SliceDataPtr->LVA_DecreaseDepth
                lgui_displayFrame, dataPtr
                lgui_updateLabels, dataPtr
        endif
        if(event.ch eq 116) then begin                                          ;t
                (*dataPtr).SliceDataPtr->LVA_IncreaseTime
                lgui_displayFrame, dataPtr
                lgui_updateLabels, dataPtr
        endif
        if(event.ch eq 117) then begin                                          ;u
                (*dataPtr).SliceDataPtr->LVA_IncreaseDepth
                lgui_displayFrame, dataPtr
                lgui_updateLabels, dataPtr
        endif
        if(event.ch eq 105) then begin
                (*dataPtr).SliceDataPtr->LVA_SetTimeIndex, 8
                lgui_displayFrame, dataPtr
                lgui_updateLabels, dataPtr
        endif
endif

if(event.type eq 0 AND event.press eq 1) then begin
        lgui_reportEvent, dataPtr, event.x, event.y
        lgui_displayFrame, dataPtr
endif
```

```
if(event.type eq 7) then begin
       if(event.clicks gt 0) then (*dataPtr).SliceDataPtr->LVA_IncreaseDepth else
(*dataPtr).SliceDataPtr->LVA DecreaseDepth
       widget control, (*dataPtr).depthSlider, SET_VALUE = (*dataPtr).SliceDataPtr-
>LVA GetDepthIndex()
       lgui displayFrame, dataPtr
       lgui_updateLabels, dataPtr
endif

if(event.type eq 0 AND event.press eq 4) then begin
       WIDGET_DISPLAYCONTEXTMENU, event.ID, event.X, event.y, (*dataPtr).secondaryContextBase
endif
endcase

'plotWindow' : begin
(*dataPtr).activeWindow = 'plotWindow'
if(event.type eq 0 AND event.press eq 4) then begin
       WIDGET_DISPLAYCONTEXTMENU, event.ID, event.X, event.y, (*dataPtr).contextMenuBase
endif
endcase

'slide depth'       : (*dataPtr).SliceDataPtr->LVA SetDepthIndex, event.value
'slide time'        : (*dataPtr).SliceDataPtr->LVA_SetTimeIndex, event.value
'slide petmin val'  : (*dataPtr).SliceDataPtr->LVA_SetMinDisplayValue, event.value, 0
'slide petmax val'  : (*dataPtr).SliceDataPtr->LVA_SetMaxDisplayValue, event.value, 0
'slide K1min val'   : (*dataPtr).SliceDataPtr->LVA SetMinDisplayValue, event.value, 1
'slide K1max val'   : (*dataPtr).SliceDataPtr->LVA SetMaxDisplayValue, event.value, 1
'slide k2min val'   : (*dataPtr).SliceDataPtr->LVA SetMinDisplayValue, event.value, 2
'slide k2max val'   : (*dataPtr).SliceDataPtr->LVA_SetMaxDisplayValue, event.value, 2
'slide k3min val'   : (*dataPtr).SliceDataPtr->LVA_SetMinDisplayValue, event.value, 3
'slide k3max val'   : (*dataPtr).SliceDataPtr->LVA_SetMaxDisplayValue, event.value, 3
'slide mbmin val'   : (*dataPtr).SliceDataPtr->LVA SetMinDisplayValue, event.value, 4
'slide mbmax val'   : (*dataPtr).SliceDataPtr->LVA SetMaxDisplayValue, event.value, 4
'slide vbmin val'   : (*dataPtr).SliceDataPtr->LVA SetMinDisplayValue, event.value, 5
'slide vbmax val'   : (*dataPtr).SliceDataPtr->LVA_SetMaxDisplayValue, event.value, 5
'slide psmin val'   : (*dataPtr).SliceDataPtr->LVA_SetMinDisplayValue, event.value, 6
'slide psmax val'   : (*dataPtr).SliceDataPtr->LVA_SetMaxDisplayValue, event.value, 6
'slide pimin val'   : (*dataPtr).SliceDataPtr->LVA SetMinDisplayValue, event.value, 7
'slide pimax val'   : (*dataPtr).SliceDataPtr->LVA SetMaxDisplayValue, event.value, 7
'slide ctmin val'   : (*dataPtr).SliceDataPtr->LVA SetMinDisplayValue, event.value, 8
'slide ctmax val'   : (*dataPtr).SliceDataPtr->LVA_SetMaxDisplayValue, event.value, 8

'type droplist' : (*dataPtr).selectType = event.index
endcase

lgui_displayFrame, dataPtr
lgui_updateLabels, dataPtr
END

PRO lgui_contextBarEvent, event
widget_control, event.top, get_uvalue=dataPtr
widget_control, event.id, get_uvalue=widget
case widget of
'Display frame'         : (*dataPtr).secondWinDisplayMode = 1
'Display CT frame'      : (*dataPtr).secondWinDisplayMode = 10
'Display blended frame' : (*dataPtr).secondWinDisplaymode = 11
'Display k1 blended frame' :(*dataPtr).secondWinDisplaymode =16
'Display k2 blended frame' :(*dataPtr).secondWinDisplaymode =17
'Display k3 blended frame' :(*dataPtr).secondWinDisplaymode =18
'Display vb blended frame' :(*dataPtr).secondWinDisplaymode =19
'Display mb blended frame' :(*dataPtr).secondWinDisplaymode = 21
'Display pSlope blended frame' :(*dataPtr).secondWinDisplaymode = 22
'Display pInter blended frame' :(*dataPtr).secondWinDisplaymode = 23
'Display histogram'     : (*dataPtr).secondWinDisplayMode = 2
'Display plasma'        : (*dataPtr).secondWinDisplayMode = 3
'Display tumor plot'    : (*dataPtr).secondWinDisplayMode = 4
'Display k1 frame'      : (*dataPtr).secondWinDisplayMode = 5
'Display k2 frame'      : (*dataPtr).secondWinDisplayMode = 6
'Display k3 frame'      : (*dataPtr).secondWinDisplayMode = 7
'Display k4 frame'      : (*dataPtr).secondWinDisplayMode = 8
'Display vb frame'      : (*dataPtr).secondWinDisplayMode = 13
'Display r2 frame'      : (*dataPtr).secondWinDisplayMode = 20
'Display voxel function': (*dataPtr).secondWinDisplayMode = 9
'Display tissue plot'   : (*dataPtr).secondWinDisplayMode = 12
'Display tumor tissue ratio': (*dataPtr).secondWinDisplayMode = 14
'Display patlak'        : (*dataPtr).secondWinDisplayMode = 15
```

```
'Display sec frame'     : (*dataPtr).primaryWinDisplayMode = 1
'Display sec CT frame'  : (*dataPtr).primaryWinDisplayMode = 10
'Display sec blended frame' : (*dataPtr).primaryWinDisplayMode = 11
'Display sec k1 blended frame' :(*dataPtr).primaryWinDisplayMode =16
'Display sec k2 blended frame' :(*dataPtr).primaryWinDisplayMode =17
'Display sec k3 blended frame' :(*dataPtr).primaryWinDisplayMode =18
'Display sec vb blended frame' :(*dataPtr).primaryWinDisplayMode =19
'Display sec mb blended frame' :(*dataPtr).primaryWinDisplayMode = 21
'Display sec pSlope blend' :(*dataPtr).primaryWinDisplayMode = 22
'Display sec pInter blend' :(*dataPtr).primaryWinDisplayMode = 23

'Display sec plasma'    : (*dataPtr).primaryWinDisplayMode = 3
'Display sec k1'        : (*dataPtr).primaryWinDisplayMode = 5
'Display sec k2'        : (*dataPtr).primaryWinDisplayMode = 6
'Display sec k3'        : (*dataPtr).primaryWinDisplayMode = 7
'Display sec r2'      : (*dataPtr).primaryWinDisplayMode = 20
'Define region'         : begin
        COMMON T, draw
        WSET, (*dataPtr).plotWinid
        q= CW_DEFROI(draw, /RESTORE)
        if (q[0] ne -1) then begin
         ;print, size(q)
         mat = array_indices(indgen(512,512),q)
         downmat = mat / 8 ;trunkate it
         restoremat = downmat*8
         indexmat = transpose(restoremat)#[1, 512]
         sortmat = indexmat[sort(indexmat)]
         uniqmat = sortmat(uniq(sortmat))
               qlength = (size(uniqmat, /dimensions))[0]
               ;print, qlength
               ;stop
               (*dataPtr).SliceDataPtr->LVA SetReportAddOnly, 1B
               for index = 0, qlength-1 do begin
                       rows = uniqmat[index] / 512
                       cols = uniqmat[index] - rows*512
                       lgui_reportEvent, dataPtr, cols, rows
               endfor
               (*dataPtr).SliceDataPtr->LVA_SetReportAddOnly, 0B
               lgui_displayFrame, dataPtr
        endif
endcase

else: return
endcase
lgui_displayFrame, dataPtr
END

PRO lgui_displayPreferences, event
widget control, event.top, get uvalue=dataPtr
widget_control, event.id, get_uvalue=widget
varname = PropertiesBox(Title='Properties', Group_Leader=event.top, Cancel=cancelled,
XSize=100, dataPtr=dataPtr)
END

PRO lgui_reportEvent, dataPtr, xpos, ypos
case (*dataPtr).selectType of
1 : (*dataPtr).SliceDataPtr->LVA_ReportPlasmaPoint, xpos, ypos
2 : (*dataPtr).SliceDataPtr->LVA ReportTumorPoint, xpos, ypos
3 : (*dataPtr).SliceDataPtr->LVA ReportTissuePoint, xpos, ypos
4 : (*dataPtr).SliceDataPtr->LVA ReportSelectedPoint, xpos, ypos
5 : (*dataPtr).SliceDataPtr->LVA_ReportCenterPoint, xpos, ypos
6 : begin
        if((*dataPtr).rectangleFirstPoint eq 1) then begin
               (*dataPtr).rectangleFirstPoint = 0
               (*dataPtr).rectangleXPoint = xpos
               (*dataPtr).rectangleYPoint = ypos
        endif else begin
               (*dataPtr).rectangleFirstPoint = 1
               (*dataPtr).SliceDataPtr->LVA SetTumorOutputWindow, xpos, ypos,
(*dataPtr).rectangleXPoint, (*dataPtr).rectangleYPoint
        endelse
endcase
7 : begin
        if((*dataPtr).rectangleFirstPoint eq 1) then begin
               (*dataPtr).rectangleFirstPoint = 0
```

```
                (*dataPtr).rectangleXPoint = xpos
                (*dataPtr).rectangleYPoint = ypos
        endif else begin
                (*dataPtr).rectangleFirstPoint = 1
                (*dataPtr).SliceDataPtr->LVA SetOutputWindow, xpos, ypos,
(*dataPtr).rectangleXPoint, (*dataPtr).rectangleYPoint
        endelse
endcase
8 : (*dataPtr).SliceDataPtr->LVA SetTextPosition, xpos, ypos
9 : (*dataPtr).SliceDataPtr->LVA_SetLegendPosition, xpos, ypos
else :
endcase
END

PRO lgui_break, EVENT
widget_control, EVENT.top, get_uvalue=dataPtr
stop
;(*dataPtr).SliceDataPtr->LVA_SetManualCTShift, 10, 1
END

PRO lgui_LVAbreak, EVENT
widget control, EVENT.top, get uvalue=dataPtr
(*dataPtr).SliceDataPtr->LVA_Break
END

PRO lgui_updateLabels, dataPtr
widget control, (*dataPtr).depthLabel, set_value = string((*dataPtr).SliceDataPtr-
>LVA_GetDepth(), format = '(I5)')
time = (*dataPtr).SliceDataPtr->LVA_GetTime()- (*dataPtr).SliceDataPtr->LVA_GetStartTime()
minutes = floor(time*60.+0.0001)
mysec = round((time*60. - minutes)*60.)
timestring = string(minutes, format = '(I2)')+':'+ string(mysec, format = '(I2)')
widget control, (*dataPtr).timeLabel, set value = timestring
PETstr = string((*dataPtr).SliceDataPtr->LVA_GetMinDisplayLabel(0), format= '(F5.2)')+ ':' +
string((*dataPtr).SliceDataPtr->LVA_GetMaxDisplayLabel(0), format= '(F5.2)')
widget_control, (*dataPtr).PETLabel, set_value = PETstr
K1str = string((*dataPtr).SliceDataPtr->LVA_GetMinDisplayLabel(1), format= '(F5.2)')+ ':' +
string((*dataPtr).SliceDataPtr->LVA_GetMaxDisplayLabel(1), format= '(F5.2)')
widget control, (*dataPtr).K1Label, set value = K1str
k2str = string((*dataPtr).SliceDataPtr->LVA_GetMinDisplayLabel(2), format= '(F5.2)')+ ':' +
string((*dataPtr).SliceDataPtr->LVA_GetMaxDisplayLabel(2), format= '(F5.2)')
widget_control, (*dataPtr).k2Label, set_value = k2str
k3str = string((*dataPtr).SliceDataPtr->LVA_GetMinDisplayLabel(3), format= '(F5.2)')+ ':' +
string((*dataPtr).SliceDataPtr->LVA_GetMaxDisplayLabel(3), format= '(F5.2)')
widget_control, (*dataPtr).k3Label, set_value = k3str
psstr = string((*dataPtr).SliceDataPtr->LVA_GetMinDisplayLabel(6), format= '(F5.2)')+ ':' +
string((*dataPtr).SliceDataPtr->LVA_GetMaxDisplayLabel(6), format= '(F5.2)')
widget_control, (*dataPtr).psLabel, set_value = psstr
pistr = string((*dataPtr).SliceDataPtr->LVA_GetMinDisplayLabel(7), format= '(F5.2)')+ ':' +
string((*dataPtr).SliceDataPtr->LVA_GetMaxDisplayLabel(7), format= '(F5.2)')
widget_control, (*dataPtr).piLabel, set_value = pistr
mbstr = string((*dataPtr).SliceDataPtr->LVA_GetMinDisplayLabel(4), format= '(F5.2)')+ ':' +
string((*dataPtr).SliceDataPtr->LVA_GetMaxDisplayLabel(4), format= '(F5.2)')
widget_control, (*dataPtr).mbLabel, set_value = mbstr
vbstr = string((*dataPtr).SliceDataPtr->LVA_GetMinDisplayLabel(5), format= '(F5.2)')+ ':' +
string((*dataPtr).SliceDataPtr->LVA_GetMaxDisplayLabel(5), format= '(F5.2)')
widget_control, (*dataPtr).vbLabel, set_value = vbstr
ctstr = string((*dataPtr).SliceDataPtr->LVA_GetMinDisplayLabel(8), format= '(F5.2)')+ ':' +
string((*dataPtr).SliceDataPtr->LVA_GetMaxDisplayLabel(8), format= '(F5.2)')
widget_control, (*dataPtr).CTLabel, set_value = ctstr
END

PRO lgui_updateSliders, dataPtr
;Min sliders
PETval = (*dataPtr).SliceDataPtr->LVA GetMinDisplayValue(0)
widget control, (*dataPtr).minPetSlider, set value = PETval
K1val = (*dataPtr).SliceDataPtr->LVA_GetMinDisplayValue(1)
widget_control, (*dataPtr).minK1Slider, set_value = K1val
k2val = (*dataPtr).SliceDataPtr->LVA_GetMinDisplayValue(2)
widget control, (*dataPtr).mink2Slider, set value = k2val
k3val = (*dataPtr).SliceDataPtr->LVA GetMinDisplayValue(3)
widget control, (*dataPtr).mink3Slider, set value = k3val
mbval = (*dataPtr).SliceDataPtr->LVA_GetMinDisplayValue(4)
widget_control, (*dataPtr).minMbSlider, set_value = mbval
vbval = (*dataPtr).SliceDataPtr->LVA_GetMinDisplayValue(5)
widget_control, (*dataPtr).minVbSlider, set_value = vbval
```

```
psval = (*dataPtr).SliceDataPtr->LVA_GetMinDisplayValue(6)
widget_control, (*dataPtr).minpsSlider, set_value = psval
pival = (*dataPtr).SliceDataPtr->LVA GetMinDisplayValue(7)
widget control, (*dataPtr).minpiSlider, set value = pival
ctval = (*dataPtr).SliceDataPtr->LVA GetMinDisplayValue(8)
widget_control, (*dataPtr).minCTSlider, set_value = ctval

;Max sliders
PETval = (*dataPtr).SliceDataPtr->LVA GetMaxDisplayValue(0)
widget control, (*dataPtr).maxPetSlider, set value = PETval
K1val = (*dataPtr).SliceDataPtr->LVA GetMaxDisplayValue(1)
widget_control, (*dataPtr).maxK1Slider, set_value = K1val
k2val = (*dataPtr).SliceDataPtr->LVA_GetMaxDisplayValue(2)
widget_control, (*dataPtr).maxk2Slider, set_value = k2val
k3val = (*dataPtr).SliceDataPtr->LVA GetMaxDisplayValue(3)
widget control, (*dataPtr).maxk3Slider, set value = k3val
mbval = (*dataPtr).SliceDataPtr->LVA_GetMaxDisplayValue(4)
widget_control, (*dataPtr).maxMbSlider, set_value = mbval
vbval = (*dataPtr).SliceDataPtr->LVA_GetMaxDisplayValue(5)
widget control, (*dataPtr).maxVbSlider, set value = vbval
psval = (*dataPtr).SliceDataPtr->LVA GetMaxDisplayValue(6)
widget control, (*dataPtr).maxpsSlider, set value = psval
pival = (*dataPtr).SliceDataPtr->LVA_GetMaxDisplayValue(7)
widget_control, (*dataPtr).maxpiSlider, set_value = pival
ctval = (*dataPtr).SliceDataPtr->LVA_GetMaxDisplayValue(8)
widget_control, (*dataPtr).maxCTSlider, set_value = ctval
END

;Basic load of all dicom images in a directory and store it as the PET data
PRO lgui_loadDirectory, EVENT
widget_control, EVENT.top, get_uvalue=dataPtr
directoryPath = DIALOG_PICKFILE(/READ, /DIRECTORY, TITLE = 'Select DICOM directory for PET
images')
(*dataPtr).sliceDataPtr->LVA_LoadDirectory, directoryPath

widget_control, (*dataPtr).depthSlider, SET_SLIDER_MAX = (*dataPtr).sliceDataPtr-
>LVA_getMaxDepthIndex()
widget control, (*dataPtr).timeSlider, SET_SLIDER_MAX = (*dataPtr).sliceDataPtr-
>LVA getMaxTimeIndex()
lgui displayFrame, dataPtr
lgui_updateLabels, dataPtr

END

;load all CT images
PRO lgui_loadCTDirectory, EVENT
widget_control, EVENT.top, get_uvalue=dataPtr
directoryPath = DIALOG_PICKFILE(/READ, /DIRECTORY, TITLE = 'Select DICOM directory for CT
images')
(*dataPtr).sliceDataPtr->LVA_LoadCTDirectory, directoryPath

lgui_displayFrame, dataPtr
END

;files = DIALOG PICKFILE(/READ, /MULTIPLE_FILES, GET_PATH = directoryPath, TITLE = 'Select
DICOM files', FILTER='*.dcm')

;The load file procedure, loading a lva file
PRO lgui_load, EVENT
widget_control, EVENT.top, get_uvalue=dataPtr

if( (*dataPtr).SliceDataPtr->LVA_DebugMode() eq 1) then begin
(*dataPtr).SliceDataPtr->LVA_LoadState, 'G:\SRC\pas2h.lva'
endif else begin
path = DIALOG_PICKFILE(filter='*.lva', /READ, TITLE = 'Load lva file')
(*dataPtr).SliceDataPtr->LVA LoadState, path
(*dataPtr).SliceDataPtr->LVA_CalculateKSpace, 1
endElse
widget_control, (*dataPtr).depthSlider, SET_SLIDER_MAX = (*dataPtr).sliceDataPtr-
>LVA getMaxDepthIndex()
widget control, (*dataPtr).timeSlider, SET_SLIDER_MAX = (*dataPtr).sliceDataPtr-
>LVA getMaxTimeIndex()
;widget_control, EVENT.top, set_value = 'title'
lgui_displayFrame, dataPtr
lgui_updateLabels, dataPtr
lgui_updateSliders, dataPtr
```

```
END

PRO lgui_save, event
widget control, EVENT.top, get uvalue=dataPtr
path = DIALOG PICKFILE(default_extension='lva', filter='*.lva', /WRITE, TITLE = 'Save to lva
file')
(*dataPtr).SliceDataPtr->LVA_SaveState, path
END

PRO lgui_export, event
widget control, EVENT.top, get uvalue=dataPtr
savefile = DIALOG PICKFILE(default_extension='trc', filter='*.trc', /WRITE, TITLE = 'Save to
trc file')
(*dataPtr).SliceDataPtr->LVA_SaveTrace, savefile
END

PRO lgui_calculateCurve, EVENT
widget_control, EVENT.top, get_uvalue=dataPtr
widget_control, EVENT.id, get_uvalue=widget
case widget of
'CalcCurveSmoothed' : (*dataPtr).SliceDataPtr->LVA CalculateKSpace, 1
'CalcCurveSmoothedFrame' : (*dataPtr).SliceDataPtr->LVA CalculateKSpace, 1, 1
'CalcCurve' : (*dataPtr).SliceDataPtr->LVA_CalculateKSpace
'CalcCurveFrame' : (*dataPtr).SliceDataPtr->LVA_CalculateKSpace, 0, 1
'CalcPearsons' : (*dataPtr).SliceDataPtr->LVA_CalculateR2Map
'CalcPearsonsFrame' : (*dataPtr).SliceDataPtr->LVA CalculateR2Map, 1
'CalcPatlakFrame' : (*dataPtr).SliceDataPtr->LVA_CalculatePatlakMap,1
endcase
lgui_displayFrame, dataPtr
END

PRO lgui_resetPlasma, EVENT
widget control, EVENT.top, get uvalue=dataPtr
(*dataPtr).SliceDataPtr->LVA_ClearPlasmaPoints
lgui_displayFrame, dataPtr
END

PRO lgui_resetTumor, EVENT
widget control, EVENT.top, get uvalue=dataPtr
(*dataPtr).SliceDataPtr->LVA_ClearTumorPoints
lgui_displayFrame, dataPtr
END

PRO lgui_togglePlasma, EVENT
widget_control, EVENT.top, get_uvalue=dataPtr

if((*dataPtr).displayPlasma eq 0L) then begin (*dataPtr).displayPlasma = 1L
endif else begin (*dataPtr).displayPlasma = 0L
endelse
lgui_displayFrame, dataPtr
END

PRO lgui_displayTumorROI, EVENT
widget_control, EVENT.top, get_uvalue=dataPtr
(*dataPtr).sliceDataPtr->LVA_ToggleTumorROI
lgui_displayFrame, dataPtr
END

PRO lgui_displayTumorEdgeROI, EVENT
widget control, EVENT.top, get uvalue=dataPtr
(*dataPtr).sliceDataPtr->LVA ToggleTumorEdge
if((*dataPtr).tumorEdge eq 1B) then (*dataPtr).tumorEdge = 0B else (*dataPtr).tumorEdge = 1B
lgui_displayFrame, dataPtr
END

PRO lgui_displayPlasmaROI, EVENT
widget_control, EVENT.top, get_uvalue=dataPtr
(*dataPtr).sliceDataPtr->LVA_TogglePlasmaROI
lgui_displayFrame, dataPtr
END

PRO lgui_displayTissueROI, EVENT
widget_control, EVENT.top, get_uvalue=dataPtr
(*dataPtr).sliceDataPtr->LVA_ToggleTissueROI
lgui_displayFrame, dataPtr
END
```

```
PRO lgui_toggleZoom, EVENT
widget control, EVENT.top, get uvalue=dataPtr
(*dataPtr).sliceDataPtr->LVA_ToggleZoom
lgui_displayFrame, dataPtr
END

PRO lgui_togglePETSmoothing, EVENT
widget control, EVENT.top, get uvalue=dataPtr
(*dataPtr).sliceDataPtr->LVA_TogglePetSmoothing
lgui_displayFrame, dataPtr
END


PRO lgui_toggleTumorROISmoothing, EVENT
widget control, EVENT.top, get uvalue=dataPtr
if((*dataPtr).tumorROISmoothing eq 0L) then (*dataPtr).tumorROISmoothing = 9L $
else (*dataPtr).tumorROISmoothing = 0L
lgui_displayFrame, dataPtr
END

PRO lgui_toggleTumorWindow, EVENT
widget_control, EVENT.top, get_uvalue=dataPtr
(*dataPtr).sliceDataPtr->LVA_ToggleTumorWindow
lgui_displayFrame, dataPtr
END

PRO lgui_toggleOutputWindow, EVENT
widget_control, EVENT.top, get_uvalue=dataPtr
(*dataPtr).sliceDataPtr->LVA_ToggleOutputWindow
lgui_displayFrame, dataPtr
END

;The display function
PRO lgui_displayFrame, dataPtr

if(ptr_valid(dataPtr) eq 0) then return
;Display nr 1
lgui drawFrame, dataPtr, (*dataPtr).primaryWinDisplayMode, (*dataPtr).renderWinid
wset, (*dataPtr).outputWinid
Device, Copy= [0,0,512,512,0,0, (*dataPtr).renderWinid]

;Display nr 2
lgui drawFrame, dataPtr, (*dataPtr).secondWinDisplayMode, (*dataPtr).renderWinid
wset, (*dataPtr).plotWinid
Device, Copy= [0,0,512,512,0,0, (*dataPtr).renderWinid]
END

PRO lgui_drawFrame, dataPtr, drawmode, activeWindow
wset, activeWindow
;Draw frame

if(drawMode eq 1) then begin
if((*dataPtr).tumorROISmoothing eq 0) then begin
(*dataPtr).SliceDataPtr->LVA DisplaySlice
if((*dataPtr).displayPlasma eq 1L) then (*dataPtr).SliceDataPtr->LVA_DisplayPlasma
(*dataPtr).SliceDataPtr->LVA_DisplayTumor
(*dataPtr).SliceDataPtr->LVA_DisplayTissue
(*dataPtr).SliceDataPtr->LVA DisplaySelectedPixel
(*dataPtr).SliceDataPtr->LVA DisplayTumorWindow
(*dataPtr).SliceDataPtr->LVA DisplayOutputWindow
if((*dataPtr).tumorEdge eq 1) then (*dataPtr).SliceDataPtr->LVA_BlendEdgeTumorOntoFrame, 9,
activeWindow
endif else begin
(*dataPtr).SliceDataPtr->LVA DisplaySlice
if((*dataPtr).displayPlasma eq 1L) then (*dataPtr).SliceDataPtr->LVA_DisplayPlasma
(*dataPtr).SliceDataPtr->LVA_DisplayTissue
(*dataPtr).SliceDataPtr->LVA_DisplaySelectedPixel
(*dataPtr).SliceDataPtr->LVA_DisplayTumorWindow
(*dataPtr).SliceDataPtr->LVA DisplayOutputWindow
if((*dataPtr).tumorEdge eq 1) then (*dataPtr).SliceDataPtr->LVA_BlendEdgeTumorOntoFrame, 9,
activeWindow

;if((size(tumorFrame)[2] gt )
  baseFrame = TVREAD(TRUE=3)
```

```
  Window, XSIZE = 512, YSIZE = 512, /FREE, /PIXMAP
  pixmapWin = !D.Window
  WSet, pixmapWin
  (*dataPtr).SliceDataPtr->LVA_DisplayContinousTumor
  tumorFrame = TVREAD(TRUE=3)
  WDelete, pixmapWin
  wset, activeWindow
  alpha = 0.3
  maxVal = max(baseFrame)
  if(size(size(tumorFrame, /dimensions), /dimensions) eq 3) then begin
        redSmoothed = filter image( tumorFrame[*,*,0], SMOOTH=(*dataPtr).tumorROISmoothing,
        MEDIAN=(*dataPtr).tumorROISmoothing)
        greenSmoothed = filter_image( tumorFrame[*,*,1], SMOOTH=(*dataPtr).tumorROISmoothing,
        MEDIAN=(*dataPtr).tumorROISmoothing)
        blueSmoothed = filter image( tumorFrame[*,*,2], SMOOTH=(*dataPtr).tumorROISmoothing,
        MEDIAN=(*dataPtr).tumorROISmoothing)
        tumorFrame[*,*,0] = redSmoothed
        tumorFrame[*,*,1] = greenSmoothed
        tumorFrame[*,*,2] = blueSmoothed
        addframe = (1-alpha)*tumorFrame
        overflow = where(replicate(maxVal, 512, 512, 3)-addframe lt baseFrame)
        blendframe = baseFrame + addframe
        if((size(overflow, /dimensions))[0] gt 0) then blendframe[overflow] = maxVal
        TV, blendframe, TRUE=3
 endif else begin
        tumorFrame = filter image( tumorFrame, SMOOTH=(*dataPtr).tumorROISmoothing,
        MEDIAN=(*dataPtr).tumorROISmoothing)
        addframe = (1-alpha)*tumorFrame
        overflow = where(replicate(maxVal, 512, 512)-addframe lt baseFrame)
        blendframe = baseFrame + addframe
        if((size(overflow, /dimensions))[0] gt 0) then blendframe[overflow] = maxVal
        TV, blendframe
 endelse
endelse
endif

;Draw frame histogram
if(drawMode eq 2) then (*dataPtr).sliceDataPtr->LVA_DisplayHistogram

;Draw plasma intensity history curve
if(drawMode eq 3) then (*dataPtr).SliceDataPtr->LVA_PlotPlasma

;Draw tumor intensity history curve
if(drawMode eq 4) then (*dataPtr).SliceDataPtr->LVA_PlotTumor

if(drawMode eq 5) then (*dataPtr).SliceDataPtr->LVA_DisplayKSlice, 1
if(drawMode eq 6) then (*dataPtr).SliceDataPtr->LVA_DisplayKSlice, 2
if(drawMode eq 7) then (*dataPtr).SliceDataPtr->LVA_DisplayKSlice, 3
if(drawMode eq 8) then (*dataPtr).SliceDataPtr->LVA_DisplayKSlice, 4
if(drawMode eq 9) then (*dataPtr).SliceDataPtr->LVA_PlotSelectedVoxel
if(drawMode eq 10) then begin
if((*dataPtr).tumorROISmoothing eq 0) then begin
        (*dataPtr).SliceDataPtr->LVA_DisplayCTSlice
        if((*dataPtr).displayPlasma eq 1L) then (*dataPtr).SliceDataPtr->LVA_DisplayPlasma
        (*dataPtr).SliceDataPtr->LVA DisplayTumor
        (*dataPtr).SliceDataPtr->LVA DisplayTissue
        (*dataPtr).SliceDataPtr->LVA_DisplaySelectedPixel
        (*dataPtr).SliceDataPtr->LVA_DisplayTumorWindow
(*dataPtr).SliceDataPtr->LVA_DisplayOutputWindow
endif else begin
        (*dataPtr).SliceDataPtr->LVA DisplayCTSlice
        if((*dataPtr).displayPlasma eq 1L) then (*dataPtr).SliceDataPtr->LVA_DisplayPlasma
        (*dataPtr).SliceDataPtr->LVA_DisplayTissue
        (*dataPtr).SliceDataPtr->LVA_DisplaySelectedPixel
        (*dataPtr).SliceDataPtr->LVA DisplayTumorWindow
        (*dataPtr).SliceDataPtr->LVA_DisplayOutputWindow

        baseFrame = TVREAD(TRUE=3)

        Window, XSIZE = 512, YSIZE = 512, /FREE, /PIXMAP
        pixmapWin = !D.Window
        WSet, pixmapWin
        (*dataPtr).SliceDataPtr->LVA_DisplayContinousTumor
        tumorFrame = TVREAD(TRUE=3)
        WDelete, pixmapWin
        wset, activeWindow
```

```
        alpha = 0.8
        maxVal = max(baseFrame)
        if(size(size(tumorFrame, /dimensions), /dimensions) eq 3) then begin
          redSmoothed = filter image( tumorFrame[*,*,0], SMOOTH=(*dataPtr).tumorROISmoothing,
        MEDIAN=(*dataPtr).tumorROISmoothing)
          greenSmoothed = filter image( tumorFrame[*,*,1], SMOOTH=(*dataPtr).tumorROISmoothing,
               MEDIAN=(*dataPtr).tumorROISmoothing)
          blueSmoothed = filter_image( tumorFrame[*,*,2], SMOOTH=(*dataPtr).tumorROISmoothing,
               MEDIAN=(*dataPtr).tumorROISmoothing)
          tumorFrame[*,*,0] = redSmoothed
          tumorFrame[*,*,1] = greenSmoothed
          tumorFrame[*,*,2] = blueSmoothed
          addframe = (0.2)*tumorFrame
          overflow = where(replicate(maxVal, 512, 512, 3)-addframe lt baseFrame)
          blendframe = baseFrame + addframe
          if((size(overflow, /dimensions))[0] gt 0) then blendframe[overflow] = maxVal
          TV, blendframe, TRUE=3
        endif else begin
          tumorFrame = filter_image( tumorFrame, SMOOTH=(*dataPtr).tumorROISmoothing,
               MEDIAN=(*dataPtr).tumorROISmoothing)
          addframe = (1-alpha)*tumorFrame
          overflow = where(replicate(maxVal, 512, 512)-addframe lt baseFrame)
          blendframe = baseFrame + addframe
          if((size(overflow, /dimensions))[0] gt 0) then blendframe[overflow] = maxVal
          TV, blendframe
        endelse

endelse
endif
if(drawmode eq 11) then (*dataPtr).SliceDataPtr->LVA_DisplaySUVBlended
if(drawMode eq 12) then (*dataPtr).SliceDataPtr->LVA_PlotTissue
if(drawMode eq 13) then (*dataPtr).SliceDataPtr->LVA_DisplayVBSlice
if(drawMode eq 14) then (*dataPtr).SliceDataPtr->LVA_PlotTumorPlasmaRatio
;LVA PlotTumorTissueRatio
if(drawMode eq 15) then (*dataPtr).SliceDataPtr-
>LVA_PlotSelectedPatlak;LVA_PlotTumorPlasmaRatio
if(drawMode eq 16) then (*dataPtr).SliceDataPtr->LVA_DisplayKComposite, 1
if(drawMode eq 17) then (*dataPtr).SliceDataPtr->LVA_DisplayKComposite, 2
if(drawMode eq 18) then (*dataPtr).SliceDataPtr->LVA_DisplayKComposite, 3
if(drawMode eq 19) then (*dataPtr).SliceDataPtr->LVA_DisplayVBComposite
if(drawMode eq 20) then (*dataPtr).SliceDataPtr->LVA_DisplayR2Slice
if(drawMode eq 21) then (*dataPtr).SliceDataPtr->LVA_DisplayMetabolicComposite
if(drawMode eq 22) then (*dataPtr).SliceDataPtr->LVA_DisplayPatlakSlopeSlice
if(drawMode eq 23) then (*dataPtr).SliceDataPtr->LVA_DisplayPatlakInterceptSlice
END

;I have had enough
PRO lgui_exit, EVENT
widget_control, event.top, /destroy
END

;The destructor
PRO lgui_cleanup, ID
widget_control, id, get_uvalue=dataPtr

; Clean uo the structure
;(*dataPtr).sliceDataPtr->LVA_SaveState
OBJ_DESTROY, (*dataPtr).sliceDataPtr
if(ptr_valid(dataPtr) eq 1) then ptr_free, dataPtr
print, 'Have a nice day'
END
```

The properties menu

```
;The properties popup
FUNCTION PropertiesBox, Title=title, Cancel=cancel, Group_Leader=groupleader, XSize=xsize,
dataPtr=dataPtr
cancel = 0
destoy_groupleader = 0

Catch, theError
if theError NE 0 then begin   ;Abort mission
```

```
Catch, /Cancel
ok = Dialog_Message(!Error_State.Msg)
if destoy_groupleader then Widget_Control, groupleader, /Destroy
cancel = 1
return, ""
endif

; Check parameters and keywords.
if N_Elements(title) EQ 0 then title = 'Provide Input:'
if N_Elements(xsize) EQ 0 then xsize = 200

if N_Elements(groupleader) EQ 0 then begin
groupleader = Widget_Base(Map=0)
Widget_Control, groupleader, /Realize
destroy groupleader = 1
endif else destroy_groupleader = 0
initialK  = (*dataPtr).sliceDataPtr->LVA_GetInitalKEstimates()
Klimits   = (*dataPtr).sliceDataPtr->LVA_GetKLimits()
initialVB = (*dataPtr).sliceDataPtr->LVA_GetInitialVBEstimates()
VBlimits  = (*dataPtr).sliceDataPtr->LVA GetVBLimits()
relstep   = (*dataPtr).sliceDataPtr->LVA GetRelStep()
plasmaRat = (*dataPtr).sliceDataPtr->LVA GetMinPlasmaRatio()
plasmaRatLim = (*dataPtr).sliceDataPtr->LVA_GetPlasmaRatioLimited()
tumorList  = (*dataPtr).sliceDataPtr->LVA_GetAcceptedTumorArray()
currentTumor = (*dataPtr).sliceDataPtr->LVA_GetTumorCode()
locationList  = (*dataPtr).sliceDataPtr->LVA GetAcceptedLocationArray()
currentLocation = (*dataPtr).sliceDataPtr->LVA GetLocationCode()
patientBirth  = (*dataPtr).sliceDataPtr->LVA_GetPatientBirth()

; Create modal base widget.

tlb = Widget_Base(Title=title, Column=1, /Modal, /Base_Align_Center, Group_Leader=groupleader,
uvalue='tlb')

lineBase    = Widget_Base(tlb, row = 1, uvalue='linebase')

entryHeight = 22

flcb     = Widget Base(lineBase, column = 1, uvalue='flcb')
par1label   = Widget Label(flcb, Scr YSize= entryHeight, Value='par1', uvalue='par1label')
par1label   = Widget_Label(flcb, Scr_YSize= entryHeight, Value='par2', uvalue='par2label')
par3label   = Widget_Label(flcb, Scr_YSize= entryHeight, Value='par3', uvalue='par3label')
par4label   = Widget Label(flcb, Scr YSize= entryHeight, Value='par4', uvalue='par4label')
vblabel     = Widget Label(flcb, Scr YSize= entryHeight, Value='vb   ', uvalue='vblabel')
steplabel   = Widget_Label(flcb, Scr_YSize= entryHeight, Value='step ', uvalue='steplabel')
classlabel   = Widget_Label(flcb, Scr_YSize= entryHeight, Value='Class', uvalue='harmless')

fvcb     = Widget_Base(lineBase, column = 1, uvalue='fcb')
par1textID  = Widget Text(fvcb, Scr YSize= entryHeight, /Editable, Scr_XSize=xsize,
Value=string(initialK[0]), uvalue='terminating')
par2textID  = Widget_Text(fvcb, Scr_YSize= entryHeight, /Editable, Scr_XSize=xsize,
Value=string(initialK[1]), uvalue='terminating')
par3textID  = Widget_Text(fvcb, Scr_YSize= entryHeight, /Editable, Scr_XSize=xsize,
Value=string(initialK[2]), uvalue='terminating')
par4textID  = Widget Text(fvcb, Scr YSize= entryHeight, /Editable, Scr_XSize=xsize,
Value=string(initialK[3]), uvalue='terminating')
vbtextID    = Widget_Text(fvcb, Scr_YSize= entryHeight, /Editable, Scr_XSize=xsize,
Value=string(initialvb), uvalue='terminating')
steptextID  = Widget_Text(fvcb, Scr YSize= entryHeight, /Editable, Scr_XSize=xsize,
Value=string(relstep), uvalue='terminating')
tumClasstextID = widget droplist(fvcb, Scr_YSize= entryHeight, Scr_XSize=xsize,
Value=tumorList, uvalue='harmless')
widget_control, tumClasstextID, SET_DROPLIST_SELECT = currentTumor

slcb = Widget_Base(lineBase, column = 1, uvalue='slcb')
srcb     = Widget Base(slcb, column = 1, /NonExclusive, uvalue='srcb')
lowLimitp1  = Widget_Button(srcb, Scr_YSize= entryHeight, Value='Low limit', uvalue=0D)
lowLimitp2  = Widget_Button(srcb, Scr_YSize= entryHeight, Value='Low limit', uvalue=0D)
lowLimitp3  = Widget_Button(srcb, Scr_YSize= entryHeight, Value='Low limit', uvalue=0D)
lowLimitp4  = Widget_Button(srcb, Scr YSize= entryHeight, Value='Low limit', uvalue=0D)
lowLimitvb  = Widget_Button(srcb, Scr YSize= entryHeight, Value='Low limit', uvalue=0D)
lowLimitpl  = Widget Button(srcb,  Scr YSize= entryHeight, Value='Low ratio', uvalue=0D)
tposlabel   = Widget_Label(slcb, Scr_YSize= entryHeight, Value='Tumor position',
uvalue='harmless')

if( (Klimits.k1Lim)[0] eq 1) then Widget_Control, lowLimitp1, Set_Button=1D
```

```
if( (Klimits.k2Lim)[0] eq 1) then Widget_Control, lowLimitp2, Set_Button=1D
if( (Klimits.k3Lim)[0] eq 1) then Widget_Control, lowLimitp3, Set_Button=1D
if( (Klimits.k4Lim)[0] eq 1) then Widget_Control, lowLimitp4, Set_Button=1D
if( (VBlimits.vbLim)[0] eq 1) then Widget_Control, lowLimitvb, Set_Button=1D
if( plasmaRatLim eq 1) then Widget_Control, lowLimitpl, Set_Button=1D
if( (Klimits.k1Lim)[0] eq 1) then Widget_Control, lowLimitp1, Set_Uvalue=1D
if( (Klimits.k2Lim)[0] eq 1) then Widget_Control, lowLimitp2, Set_Uvalue=1D
if( (Klimits.k3Lim)[0] eq 1) then Widget_Control, lowLimitp3, Set_Uvalue=1D
if( (Klimits.k4Lim)[0] eq 1) then Widget_Control, lowLimitp4, Set_Uvalue=1D
if( (VBlimits.vbLim)[0] eq 1) then Widget_Control, lowLimitvb, Set_Uvalue=1D
if( plasmaRatLim eq 1) then Widget_Control, lowLimitpl, Set_Uvalue=1D

svcb     = Widget_Base(lineBase, column = 1, uvalue='tcb')
low1textID  = Widget_Text(svcb, Scr_YSize= entryHeight, /Editable, Scr_XSize=xsize,
Value=string((Klimits.k1Val)[0]), uvalue='terminating')
low2textID  = Widget_Text(svcb, Scr YSize= entryHeight, /Editable, Scr_XSize=xsize,
Value=string((Klimits.k2Val)[0]), uvalue='terminating')
low3textID  = Widget_Text(svcb, Scr_YSize= entryHeight, /Editable, Scr_XSize=xsize,
Value=string((Klimits.k3Val)[0]), uvalue='terminating')
low4textID  = Widget_Text(svcb, Scr YSize= entryHeight, /Editable, Scr_XSize=xsize,
Value=string((Klimits.k4Val)[0]), uvalue='terminating')
lowvbtextID = Widget Text(svcb, Scr YSize= entryHeight, /Editable, Scr_XSize=xsize,
Value=string((VBlimits.vbVal)[0]), uvalue='terminating')
plasmatextID= Widget_Text(svcb, Scr_YSize= entryHeight, /Editable, Scr_XSize=xsize,
Value=string(plasmaRat), uvalue='terminating')
tumPostextID = widget_droplist(svcb, Scr_YSize= entryHeight, Scr_XSize=xsize,
Value=locationList, uvalue='harmless')
widget_control, tumPostextID, SET_DROPLIST_SELECT = currentLocation


tlcb = Widget_Base(lineBase, column = 1, uvalue='tlcb')
trcb    = Widget_Base(tlcb, column = 1, /NonExclusive, uvalue='trcb')
highLimitp1 = Widget Button(trcb, Scr YSize= entryHeight, Value='High limit', uvalue=0D)
highLimitp2 = Widget Button(trcb, Scr YSize= entryHeight, Value='High limit', uvalue=0D)
highLimitp3 = Widget_Button(trcb, Scr_YSize= entryHeight, Value='High limit', uvalue=0D)
highLimitp4 = Widget_Button(trcb, Scr_YSize= entryHeight, Value='High limit', uvalue=0D)
highLimitvb = Widget_Button(trcb, Scr_YSize= entryHeight, Value='High limit', uvalue=0D)
dymmylabel   = Widget Label(tlcb, Scr YSize= entryHeight, Value=' ', uvalue='harmless')
pbirlabel   = Widget_Label(tlcb, Scr_YSize= entryHeight, Value='  Patient birth  ',
uvalue='harmless')

if( (Klimits.k1Lim)[1] eq 1) then Widget_Control, highLimitp1, Set_Button=1D
if( (Klimits.k2Lim)[1] eq 1) then Widget_Control, highLimitp2, Set_Button=1D
if( (Klimits.k3Lim)[1] eq 1) then Widget_Control, highLimitp3, Set_Button=1D
if( (Klimits.k4Lim)[1] eq 1) then Widget_Control, highLimitp4, Set_Button=1D
if( (VBlimits.vbLim)[1] eq 1) then Widget_Control, highLimitvb, Set_Button=1D
if( (Klimits.k1Lim)[1] eq 1) then Widget_Control, highLimitp1, Set_Uvalue=1D
if( (Klimits.k2Lim)[1] eq 1) then Widget_Control, highLimitp2, Set_Uvalue=1D
if( (Klimits.k3Lim)[1] eq 1) then Widget_Control, highLimitp3, Set_Uvalue=1D
if( (Klimits.k4Lim)[1] eq 1) then Widget_Control, highLimitp4, Set_Uvalue=1D
if( (VBlimits.vbLim)[1] eq 1) then Widget_Control, highLimitvb, Set_Uvalue=1D

tvcb    = Widget_Base(lineBase, column = 1, uvalue='ficb')
high1textID = Widget_Text(tvcb, Scr_YSize= entryHeight, /Editable, Scr_XSize=xsize,
Value=string((Klimits.k1Val)[1]), uvalue='terminating')
high2textID = Widget Text(tvcb, Scr YSize= entryHeight, /Editable, Scr_XSize=xsize,
Value=string((Klimits.k2Val)[1]), uvalue='terminating')
high3textID = Widget_Text(tvcb, Scr_YSize= entryHeight, /Editable, Scr_XSize=xsize,
Value=string((Klimits.k3Val)[1]), uvalue='terminating')
high4textID = Widget_Text(tvcb, Scr YSize= entryHeight, /Editable, Scr_XSize=xsize,
Value=string((Klimits.k4Val)[1]), uvalue='terminating')
highvbtextID = Widget Text(tvcb, Scr YSize= entryHeight, /Editable, Scr_XSize=xsize,
Value=string((VBlimits.vbVal)[1]), uvalue='terminating')
dymmylabel   = Widget_Label(tvcb, Scr_YSize= entryHeight, Value=' ', uvalue='harmless')
birthtextID = Widget_Text(tvcb, Scr YSize= entryHeight, /Editable, Scr_XSize=xsize,
Value=string(patientBirth), uvalue='terminating')

;stepBase = Widget_Base(tlb, Row=1, uvalue='stepBase')
;stepsizelabel  = Widget_Label(stepBase, Value='Stepsize', uvalue='stepsize')
;stepSIzeID = Widget Text(stepBase, Scr YSize= 28, /Editable, Scr_XSize=xsize,
Value=string((Klimits.k1Val)[1]), uvalue='terminating')

buttonBase = Widget_Base(tlb, Row=1, uvalue='buttonBase')
cancelID = Widget_Button(buttonBase, Value='Cancel', uvalue='terminating')
acceptID = Widget_Button(buttonBase, Value='Accept', uvalue='terminating')
```

```
; Center the widgets on display.
Device, Get_Screen_Size=screenSize
IF screenSize[0] GT 2000 THEN screenSize[0] = screenSize[0]/2 ; Dual monitors.
xCenter = screenSize(0) / 2
yCenter = screenSize(1) / 2
geom = Widget Info(tlb, /Geometry)
xHalfSize = geom.Scr_XSize / 2
yHalfSize = geom.Scr_YSize / 2
Widget_Control, tlb, XOffset = xCenter-xHalfSize, YOffset = yCenter-yHalfSize

; Storage
lpropertyWidgetData = {$
par1Init:par1textID,$
par2Init:par2textID,$
par3Init:par3textID,$
par4Init:par4textID,$
vbInit:vbtextID,$
relStep:steptextID,$
lowLim1:lowLimitp1,$
lowLim2:lowLimitp2,$
lowLim3:lowLimitp3,$
lowLim4:lowLimitp4,$
lowLimvb:lowLimitvb,$
lowLimpl:lowLimitpl,$
lowVal1:low1textID,$
lowVal2:low2textID,$
lowVal3:low3textID,$
lowVal4:low4textID,$
lowValvb:lowvbtextID,$
lowMinpl:plasmatextID,$
highLim1:highLimitp1,$
highLim2:highLimitp2,$
highLim3:highLimitp3,$
highLim4:highLimitp4,$
highLimvb:highLimitvb,$
highVal1:high1textID,$
highVal2:high2textID,$
highVal3:high3textID,$
highVal4:high4textID,$
highValvb:highvbtextID,$
tumClass:tumClasstextID,$
tumPos:tumPostextID,$
birthy:birthtextID,$
GUIDataPtr:dataPtr,$
cancelID:cancelID,$
cancel:1L}

popupDataPtr = ptr_new(lpropertyWidgetData)
Widget_Control, tlb, set_uvalue=popupDataPtr

; Realize the widget hierarchy.
Widget_Control, tlb, /Realize

XManager, 'PropertiesBox', tlb

cancel = (*popupDataPtr).cancel

;Write output
Ptr Free, popupDataPtr
if destroy_groupleader then Widget_Control, groupleader, /Destroy
return, cancel
END

PRO PropertiesBox_Event, event
; This event handler responds to all events. Widget
; is always destoyed. The text is recorded if ACCEPT
; button is selected or user hits CR in text widget.
bstate = widget_info(event.id,/button_set)
widget_control, event.top, get_uvalue=dataPtr
widget_control, event.id,  get_uvalue=widget
case event.ID of
(*dataPtr).cancelID: Widget Control, event.top, /Destroy ;Cancel was hit
(*dataPtr).lowLim1: widget_control, (*dataPtr).lowLim1,
set_uvalue=widget_info(event.id,/button_set)
(*dataPtr).lowLim2: widget_control, (*dataPtr).lowLim2,
set_uvalue=widget_info(event.id,/button_set)
```

```
(*dataPtr).lowLim3: widget_control, (*dataPtr).lowLim3,
set_uvalue=widget_info(event.id,/button_set)
(*dataPtr).lowLim4: widget control, (*dataPtr).lowLim4,
set uvalue=widget info(event.id,/button set)
(*dataPtr).lowLimvb: widget control, (*dataPtr).lowLimvb,
set uvalue=widget info(event.id,/button set)
(*dataPtr).lowLimpl: widget_control, (*dataPtr).lowLimpl,
set_uvalue=widget_info(event.id,/button_set)
(*dataPtr).highLim1: widget control, (*dataPtr).highLim1,
set uvalue=widget info(event.id,/button set)
(*dataPtr).highLim2: widget control, (*dataPtr).highLim2,
set_uvalue=widget_info(event.id,/button_set)
(*dataPtr).highLim3: widget_control, (*dataPtr).highLim3,
set_uvalue=widget_info(event.id,/button_set)
(*dataPtr).highLim4: widget control, (*dataPtr).highLim4,
set uvalue=widget info(event.id,/button set)
(*dataPtr).highLimvb: widget_control, (*dataPtr).highLimvb,
set_uvalue=widget_info(event.id,/button_set)
else: begin
if(widget ne 'harmless') then begin
widget control, (*dataPtr).par1Init, get value=par1
widget control, (*dataPtr).par2Init, get value=par2
widget_control, (*dataPtr).par3Init, get_value=par3
widget_control, (*dataPtr).par4Init, get_value=par4
widget_control, (*dataPtr).vbInit, get_value=vb
widget control, (*dataPtr).relStep, get value=rs
(*((*dataPtr).GUIDataPtr)).sliceDataPtr->LVA_SetInitialKEstimates, double(par1), double(par2),
double(par3), double(par4)
(*((*dataPtr).GUIDataPtr)).sliceDataPtr->LVA_SetInitialVBEstimates, double(vb)
(*((*dataPtr).GUIDataPtr)).sliceDataPtr->LVA_SetRelstep, double(rs)

widget control, (*dataPtr).lowLim1, get uvalue=loLim1
widget control, (*dataPtr).lowLim2, get uvalue=loLim2
widget control, (*dataPtr).lowLim3, get uvalue=loLim3
widget_control, (*dataPtr).lowLim4, get_uvalue=loLim4
widget_control, (*dataPtr).lowLimvb, get_uvalue=loLimvb
widget_control, (*dataPtr).lowLimpl, get_uvalue=loLimpl
widget control, (*dataPtr).highLim1, get uvalue=hiLim1
widget control, (*dataPtr).highLim2, get uvalue=hiLim2
widget control, (*dataPtr).highLim3, get uvalue=hiLim3
widget_control, (*dataPtr).highLim4, get_uvalue=hiLim4
widget_control, (*dataPtr).highLimvb, get_uvalue=hiLimvb
widget control, (*dataPtr).lowVal1, get value=loVal1
widget control, (*dataPtr).lowVal2, get value=loVal2
widget_control, (*dataPtr).lowVal3, get_value=loVal3
widget_control, (*dataPtr).lowVal4, get_value=loVal4
widget_control, (*dataPtr).lowValvb, get_value=loValvb
widget_control, (*dataPtr).lowMinpl, get_value=loMinpl
widget control, (*dataPtr).highVal1, get value=hiVal1
widget control, (*dataPtr).highVal2, get value=hiVal2
widget_control, (*dataPtr).highVal3, get_value=hiVal3
widget_control, (*dataPtr).highVal4, get_value=hiVal4
widget_control, (*dataPtr).highValvb, get_value=hiValvb
k1Limits = [double(loLim1), double(hiLim1), double(loVal1), double(hiVal1)]
k2Limits = [double(loLim2), double(hiLim2), double(loVal2), double(hiVal2)]
k3Limits = [double(loLim3), double(hiLim3), double(loVal3), double(hiVal3)]
k4Limits = [double(loLim4), double(hiLim4), double(loVal4), double(hiVal4)]
vbLimits = [double(loLimvb), double(hiLimvb), double(loValvb), double(hiValvb)]
(*((*dataPtr).GUIDataPtr)).sliceDataPtr->LVA SetKLimits, k1Limits[0:1], k1Limits[2:3],
k2Limits[0:1], k2Limits[2:3], k3Limits[0:1], k3Limits[2:3], k4Limits[0:1], k4Limits[2:3]
(*((*dataPtr).GUIDataPtr)).sliceDataPtr->LVA SetVBLimits, vbLimits[0:1], vbLimits[2:3]
(*((*dataPtr).GUIDataPtr)).sliceDataPtr->LVA SetPlasmaRatioLimited, loLimpl
(*((*dataPtr).GUIDataPtr)).sliceDataPtr->LVA_SetMinPlasmaRatio, loMinpl
tClass = widget info((*dataPtr).tumClass, /DROPLIST_SELECT)
(*((*dataPtr).GUIDataPtr)).sliceDataPtr->LVA SetTumorCode, tClass
tpos = widget info((*dataPtr).tumPos,  /DROPLIST SELECT)
(*((*dataPtr).GUIDataPtr)).sliceDataPtr->LVA_SetLocationCode, tpos
widget_control, (*dataPtr).birthy, get_value=birthy
(*((*dataPtr).GUIDataPtr)).sliceDataPtr->LVA_SetPatientBirth, birthy
(*dataPtr).cancel = 0 ; The user hit ACCEPT.
lgui displayFrame, (*dataPtr).GUIDataPtr
Widget_Control, event.top, /Destroy
endif
 endcase
endcase
END
```

Definition of the LVA_SliceData object

```
PRO LVA_SliceData__define
struct =$
{$
LVA SliceData, $
imageDirectoryPath:'',$              ;The directory of the pet images
sliceArrayPtr:ptr new(),$            ;The  pet slice data a 4d vector
petSlicePosition:fltarr(3),$
petSliceResolution:fltarr(3),$
ctImageDirectoryPath:'',$            ;The ct directory
ctSliceArrayPtr:ptr new(),$          ;The ct slice data a 3d vector
ctImageCenterPoint:fltarr(3),$       ;The center of the CT images
manualCTShift:intarr(3),$
ctImageResolution:fltarr(3),$
timeStampArrayPtr:ptr_new(),$        ;The corresponding time values to the 4rt dimension
adjustedTimeStampArrayPtr:ptr new(),$   ;Time in minutes following the injection
depthStampArrayPtr:ptr_new(),$              ;The corresponding depth values to the 3rd
dimension
ctDepthStampArrayPtr:ptr_new(),$     ;The corresponding array for the CT
bufferEmpty:1L,$                            ;Variable to check if buffers are loaded

nrPlasmaPoints:0L,$                  ;Nr of active plasma points
maxNrPlasmaPoints:500L,$             ;Max nr of plasma points (vs dynamic realocation)
plasma:fltarr(4,500),$               ;Make sure the dimensions match max nr in constructor
displayPlasmaROI:1B,$
minPlasmaRatioLimited:1B,$
minPlasmaRatio:7D,$                  ;Exclude points with worse ratio than this
validPlasmaPoints:0L,$

nrTumorPoints:0L,$                   ;Identical structure for the tumor
maxNrTumorPoints:7000L,$
tumor:fltarr(4,7000),$               ;Again note the dependency on max nr
displayTumorROI:1B,$
displayTumorEdge:0B,$
addOnly:0B,$                         ;Forces it to only add regional points
addCache:fltarr(3),$

nrTissuePoints:0L,$                  ;Nr of tissue region points
maxNrTissuePoints:4000L,$            ;Max nr of tissue points (vs dynamic realocation)
tissue:fltarr(4,4000),$              ;Make sure the dimensions match max nr in constructor
displayTissueROI:1B,$

selectedX:-1,$                       ;The selected pixel, for voxel output
selectedY:-1,$
selectedZ:-1,$

;displayRange:fltarr(2),$                   ;Range limits on the display data
petDisplayRange:fltarr(2),$          ;Range limits on the display data
K1DisplayRange:fltarr(2),$           ;Range limits on the display data
k2DisplayRange:fltarr(2),$           ;Range limits on the display data
k3DisplayRange:fltarr(2),$           ;Range limits on the display data
mbDisplayRange:fltarr(2),$           ;Range limits on the display data
vbDisplayRange:fltarr(2),$           ;Range limits on the display data
psDisplayRange:fltarr(2),$           ;Range limits on the display data
piDisplayRange:fltarr(2),$           ;Range limits on the display data
dispRngRatio:150.,$                   ;Scaling the range limits to our, SUV values
petGamma:0.025, $                     ;Gamma correction when displaying pet frames
ctDisplayRange:fltarr(2),$           ;Range limits on the CT display data
depth:0L,$                                  ;Current active depth
time:0L,$                                   ;Current active time, responses report this frame

k1Ptr:ptr new(),$                           ;K space
k2Ptr:ptr_new(),$
k3Ptr:ptr_new(),$
k4Ptr:ptr_new(),$
VBPtr:ptr new(),$
pearsonPtr:ptr new(),$        ;Adaptiation quality
patSlopePtr:ptr_new(),$
patInterPtr:ptr_new(),$

k1_init:0.1,$                ;Initial k values
k2 init:0.2,$
k3 init:0.05,$
k4_init:0.0,$
```

```
VB_init:0.123,$
k1_limited:fltarr(2),$       ;K limits for the calculation
k1_limits:fltarr(2),$
k2_limited:fltarr(2),$
k2_limits:fltarr(2),$
k3_limited:fltarr(2),$
k3_limits:fltarr(2),$
k4_limited:fltarr(2),$
k4_limits:fltarr(2),$
VB_limited:fltarr(2),$
VB_limits:fltarr(2),$
relstep:0.0,$                          ;Stepsize of the least square approximation

arteriePtr:ptr_new(),$                 ;Input function
arterieInterpolPtr:ptr_new(),$         ;Equidistant vector
currentVoxelAdaptationPtr:ptr_new(),$  ;Current voxel display buffer
timeInterpolPtr:ptr_new(),$            ;Equidistant time vector

windowSmoothSetting:1L,$               ;To smooth or not to smooth
windowDisplaySize:512L,$                        ;The size of the display
windowZoomFactor:8,$                   ;The current zoom of the window display
windowXpos:41L,$                                  ;The window position
windowYpos:32L,$
windowXOffset:41L,$                    ;The zoom position
windowYOffset:32L,$
windowTumorXpos:0L,$                   ;The tumor output window, for tumor images
windowTumorYpos:0L,$
windowTumorXsize:0L,$
windowTumorYsize:0L,$
displayTumorWindow:0B,$
windowOutputXpos:0L,$                   ;The output window, for full body images
windowOutputYpos:0L,$
windowOutputXsize:0L,$
windowOutputYsize:0L,$
displayOutputWindow:0L,$

interpolFactor:4D,$                 ;The scale we expand the time vector with to make time
linear
imageIntensityScale:0D,$                   ;A scale factor to scale the pixes to correct
units

debugMode:0L,$                                    ;To use special rules
homeMode:0L,$
averagedSUV:1L,$
locked:0L,$

tumorLocationArray:make_array(3, /STRING),$
tumorLocationCode:0L,$                 ;Tumor localization
tumorTypeArray:make array(3, /STRING),$
tumorTypeCode:0L,$                     ;Tumor type
yearOfBirth:0L,$                       ;Patient age
;sliceSpacing:0L,$                     ;The distance separating the slices
weight:0.0,$                           ;Patient weight
injectedActivity:0.0,$                 ;Activity in ....
injectedActivityConversionFactor:37.0,$ ;this eksample picoCurie, we want it in Bq
SUVConversionFactor:0.0,$              ;The SUV scaling factor

fregroundColor:0L,$
backgroundColor:16777215L,$
textPosition:intarr(2),$
legendPosition:fltarr(2)$
}
END
```

Calculation functions

```
PRO LVA_Slicedata::LVA_CalculatePlasmaFunction
if( self.nrPlasmaPoints lt 1 OR ptr_valid(self.sliceArrayPtr) eq 0) then return
;Calculate the average over the plasma points
dim_t = (size(*self.sliceArrayPtr, /dimensions))[3]
intensity = dblarr(dim_t)
ratio = dblarr(self.nrPlasmaPoints)
```

```
usedPoints = 0
for sequence = 0L, self.nrPlasmaPoints -1 do begin
peakVal =  max((*self.sliceArrayPtr)[self.plasma[0, sequence], self.plasma[1,
sequence],self.plasma[2, sequence], * ])
tailVal = mean((*self.sliceArrayPtr)[self.plasma[0, sequence], self.plasma[1,
sequence],self.plasma[2, sequence], floor((dim_t-1)*0.9D):(dim_t-1) ])
ratio[sequence] = peakVal/tailVal
if(self.minPlasmaRatioLimited eq 0 OR ratio[sequence] gt self.minPlasmaRatio) then begin
usedPoints++
for iter = 0L, dim_t-1 DO BEGIN
intensity[iter] = intensity[iter] + total((*self.sliceArrayPtr)[$
(self.plasma[0, sequence]),(self.plasma[1, sequence]), $
self.plasma[2, sequence], iter ])
endfor
endif
endfor
if(usedPoints gt 0) then intensity = intensity / double(usedPoints)
self.validPlasmaPoints = usedPoints

; Store original format artery function
*self.arteriePtr = intensity
END

PRO LVA_Slicedata::LVA_CalculatePlasmaAdaptation
self->LVA_CalculatePlasmaFunction
intensity = *self.arteriePtr
;Start the plot at zero
timeStamp = double(*self.adjustedTimeStampArrayPtr)

;Interpolate the time axis
nrTimeInterpolated  = self.interpolFactor* max(timestamp)
time interpol = dindgen(self.interpolFactor*
max(*self.adjustedTimeStampArrayPtr))/self.interpolFactor
intensity_interpol  = double(interpol(intensity, timeStamp, time_interpol))

;Create required startpoints for the curve adaption function
parinfo = replicate({value:0.0D, fixed:0, limited:[0,0], limits:[0.D,0], relstep:0.0D,
tied:''}, 4)
parinfo[0:3].limited[0] = 1
parinfo[0:3].limits[0]  = 0.D
parinfo[0:3].relstep[0]  = double(1e-4)
parinfo[2].tied = strcompress(string(max(intensity)), /REMOVE_ALL)+'-P[0]'
a_in=0.8D*max(intensity)
b_in=1.0D
c_in=0.2D*max(intensity)
d_in=0.1D
par=double([a_in, b_in, c_in, d_in])

;The start values are a bit noisy so we start at the peak
mx= max(intensity interpol, location)
subIntensity = intensity_interpol[location:*]
subTime = time_interpol[0:(nrTimeInterpolated-1-location)]

func_err=subTime*0+1.0

;Calculate the plasma function
res=MPFITFUN('arterie', subTime, subIntensity, func_err, par, PARINFO=parinfo, YFIT=func_fit,
STATUS=status, QUIET=1, MAXITER=500, FTOL=1e-6, /DOUBLE)
subArter = arterie(subTime, res)

; Add the original data points
*self.arterieInterpolPtr = intensity_interpol
; Overwrite the smoothed values
(*self.arterieInterpolPtr)[location:*] = subArter
END

PRO LVA_SliceData::LVA_CalculateKSpace, smoothed, entireFrame
if(N_ELEMENTS(smoothed) eq 0) then smoothed = 0
if(N_ELEMENTS(entireFrame) eq 0) then entireFrame = 0
if( self.nrPlasmaPoints gt 0 AND self.nrTumorPoints gt 0) then begin
;Get image dimensions
progressBar = Obj New("PROGRESSBAR")
progressBar -> Start
progressBar -> SetProperty, Text='Calculating k kmap'

imdim = size(*self.sliceArrayPtr, /dimensions)
```

```
dim_z = imdim[2]
dim_t = imdim[3]
dim_xy = imdim[0] ; Assuming equal

self->LVA CalculatePlasmaAdaptation
nrTimeInterpolated  = self.interpolFactor* max(*self.adjustedTimeStampArrayPtr)
mx= max(*self.arterieInterpolPtr, location)
time_interpol = dindgen(self.interpolFactor*
max(*self.adjustedTimeStampArrayPtr))/self.interpolFactor
timeStamp = double(*self.adjustedTimeStampArrayPtr)

;Create required start points for the 3 compartment function
parinfo = replicate({value:0.0D, fixed:0, limited:[0,0], limits:[0.D,0], relstep: 0.0D}, 5)

parinfo[0].limited = self.k1 limited
parinfo[0].limits  = self.k1 limits
parinfo[1].limited = self.k2_limited
parinfo[1].limits  = self.k2_limits
parinfo[2].limited = self.k3_limited
parinfo[2].limits  = self.k3 limits
parinfo[3].limits = self.k4 limits
parinfo[3].limited = self.k4 limited
parinfo[4].limits  = self.VB_limits
parinfo[4].limited  = self.VB_limited
parinfo[0:4].relstep = self.relstep

parinfo[3].fixed = 1

;Initial k estimates
par=double([self.k1_init, self.k2_init, self.k3_init, self.k4_init, self.VB_init])

func=dblarr (nrTimeInterpolated-location);(nrTimeInterpolated)
*self.k1Ptr=replicate(0.0d, dim xy,dim xy, dim z)
*self.k2Ptr=replicate(0.0d, dim xy,dim xy, dim z)
*self.k3Ptr=replicate(0.0d, dim_xy,dim_xy, dim_z)
*self.k4Ptr=replicate(0.0d, dim_xy,dim_xy, dim_z)
*self.VBPtr=replicate(0.0d, dim_xy,dim_xy, dim_z)
*self.pearsonPtr=replicate(0.0d, dim xy,dim xy, dim z)
*self.patSlopePtr=replicate(0.0d, dim xy,dim xy, dim z)
*self.patInterPtr=replicate(0.0d, dim_xy,dim_xy, dim_z)

func_err=time_interpol*0+1.0

count = 0.
if(entireFrame eq 1) then begin
;For all points in current slice
totalcount = float(dim_xy*dim_xy)
for xp = 0L, dim_xy -1 do begin
if progressBar -> CheckCancel() then continue
for yp = 0L, dim_xy -1 do begin
x = xp
y = yp
z = self.depth

if(smoothed eq 1 AND x gt 0 AND x lt dim_xy-1 AND y gt 0 AND y lt dim_xy-1 AND z gt 0 AND z lt
dim_z -1) then begin ;Note that we ignore possible edge problems
fr = 1;
voxelDevelopment = dblarr(1,1,1,dim_t)
voxelCylinder = (*self.sliceArrayPtr)[(x-fr):(x+fr),(y-fr):(y+fr),(z-fr):(z+fr),*]
for j = 0, dim_t-1 do begin
voxelDevelopment(j) = mean( voxelCylinder(*,*,*,j))
endfor
endif else begin
voxelDevelopment = (*self.sliceArrayPtr)[x,y,z,*]
endelse

limit = 0.4/(
self.imageIntensityScale*self.weight*1000/(self.injectedActivity/self.injectedActivityConversi
onFactor))

if(n_elements(where(voxelDevelopment gt limit)) gt 3) then begin
;Interpolate the slice data to the full sized time array
func_interpol  = double(interpol(voxelDevelopment, timeStamp, time_interpol))
;Calculate
mx= max(*self.arterieInterpolPtr, location)
subTime = time_interpol[0:(nrTimeInterpolated-1-location)]
```

```
func_err=subTime*0+1.0
ttime = subtime;time_interpol;[location:*]
tfunc = func interpol[location:*]
tplas = (*self.arterieInterpolPtr)[location:*]
treKompRes=MPFITFUN('trekomp', ttime, tfunc, func err, par, PARINFO=parinfo, YFIT=func_fit,
PERROR=perror, STATUS=status, QUIET=1, MAXITER=100, FTOL=1e-6, /DOUBLE, FUNCTARGS =
{plasmaFunc:(tplas)})
endif else treKompRes = [0,0,0,0,0]
;Store
(*self.k1Ptr)[x, y, z]=treKompRes[0]
(*self.k2Ptr)[x, y, z]=treKompRes[1]
(*self.k3Ptr)[x, y, z]=treKompRes[2]
(*self.k4Ptr)[x, y, z]=treKompRes[3]
(*self.VBPtr)[x, y, z]=treKompRes[4]
count++
if(count MOD 100 eq 1) then progressBar -> Update, count*100./totalcount
endfor
endfor
endif else begin

;For all points in our region, calculate k values
totalcount = float(self.nrTumorPoints)
for tp = 0L, self.nrTumorPoints -1 do begin
if progressBar -> CheckCancel() then continue
x = self.tumor[0,tp]
y = self.tumor[1,tp]
z = self.tumor[2,tp]

if(smoothed eq 1) then begin ;Note that we ignore possible edge problems
fr = 1;
voxelDevelopment = dblarr(1,1,1,dim_t)
voxelCylinder = (*self.sliceArrayPtr)[(x-fr):(x+fr),(y-fr):(y+fr),(z-fr):(z+fr),*]
for j = 0, dim_t-1 do begin
voxelDevelopment(j) = mean( voxelCylinder(*,*,*,j))
endfor
endif else begin
voxelDevelopment = (*self.sliceArrayPtr)[x,y,z,*]
endelse

;Interpolate the slice data to the full sized time array
func_interpol  = double(interpol(voxelDevelopment, timeStamp, time_interpol))
;Calculate
mx= max(*self.arterieInterpolPtr, location)
subTime = time interpol[0:(nrTimeInterpolated-1-location)]
func_err=subTime*0+1.0
ttime = subtime;time_interpol;[location:*]
tfunc = func_interpol[location:*]
tplas = (*self.arterieInterpolPtr)[location:*]
treKompRes=MPFITFUN('trekomp', ttime, tfunc, func err, par, PARINFO=parinfo, YFIT=func_fit,
PERROR=perror, STATUS=status, QUIET=1, MAXITER=100, FTOL=1e-6, /DOUBLE, FUNCTARGS =
{plasmaFunc:(tplas)})
;Store
(*self.k1Ptr)[x, y, z]=treKompRes[0]
(*self.k2Ptr)[x, y, z]=treKompRes[1]
(*self.k3Ptr)[x, y, z]=treKompRes[2]
(*self.k4Ptr)[x, y, z]=treKompRes[3]
(*self.VBPtr)[x, y, z]=treKompRes[4]
;                     if((*self.k3Ptr)[x, y, z] lt 1e-5) then stop
;if(CHECK_MATH() gt 0) then STOP
count++
if(count MOD 100 eq 1) then progressBar -> Update, count*100./totalcount
endfor
endelse
progressBar -> Destroy
endif

END

PRO LVA_SliceData::LVA_CalculateR2Map, entireFrame
if(ptr_valid(self.timeStampArrayPtr) eq 1 AND ptr_valid(self.k1Ptr) eq 1 ) then begin

if(N_ELEMENTS(entireFrame) eq 0) then entireFrame = 0
; To avoid all the indexing we create some temporal variables
timestamp = *self.adjustedTimeStampArrayPtr                    ;Original time values

; Find the size of the time axis and create an extended one
```

```
nrTimestamps = (size(*(self.timeStampArrayPtr), /dimension))(1)
time_interpol = dindgen(self.interpolFactor* max(timestamp))/self.interpolFactor

intensity = fltarr(nrTimestamps)
smooot = 0

if(entireFrame eq 1) then begin
;For all points in current slice
for xind = 0, (size(*self.sliceArrayPtr, /dimensions))[0] -1 do begin
for yind = 0, (size(*self.sliceArrayPtr, /dimensions))[1] -1 do begin
zind = self.depth

intensity = (*self.sliceArrayPtr)[xind, yind, zind, * ]    ;Original pixel values

k1 = float((*self.k1Ptr)[xind, yind, zind])
k2 = float((*self.k2Ptr)[xind, yind, zind])
k3 = float((*self.k3Ptr)[xind, yind, zind])
k4 = float((*self.k4Ptr)[xind, yind, zind])
vb = float((*self.vbPtr)[xind, yind, zind])

r2 = 0
correl = 0

if( k1 eq 0 AND k2 eq 0 AND k3 eq 0 AND k4 eq 0) then continue
intensity_interpol  = double(interpol(*self.arteriePtr, *self.adjustedTimeStampArrayPtr,
time interpol))
voxelAdaption = trekomp(time interpol, [k1, k2, k3, k4, vb], plasmaFunc =
*self.arterieInterpolPtr )    ;Interpolated curve
correl = (correlate(voxelAdaption, intensity))^2
(*self.pearsonPtr)[xind, yind, zind] = correl
endfor
endfor
endif

for tp = 0L, self.nrTumorPoints -1 do begin
xind = self.tumor[0,tp]
yind = self.tumor[1,tp]
zind = self.tumor[2,tp]
k1 = float((*self.k1Ptr)[xind, yind, zind])
k2 = float((*self.k2Ptr)[xind, yind, zind])
k3 = float((*self.k3Ptr)[xind, yind, zind])
k4 = float((*self.k4Ptr)[xind, yind, zind])
if( k1 eq 0 AND k2 eq 0 AND k3 eq 0 AND k4 eq 0) then continue
vb = float((*self.vbPtr)[xind, yind, zind])

fr = 1
if(smooot = 1) then begin
intensity = dblarr(1,1,1,nrTimestamps)
voxelCylinder = (*self.sliceArrayPtr)[(xind-fr):(xind+fr),(yind-fr):(yind+fr),(zind-
fr):(zind+fr),*]
for j = 0, nrTimestamps-1 do begin
intensity(j) = mean( voxelCylinder(*,*,*,j))
endfor
endif else intensity = (*self.sliceArrayPtr)[xind, yind, zind, * ]
;intensity interpol  = double(interpol(*self.arteriePtr, *self.adjustedTimeStampArrayPtr,
time interpol))
;voxelAdaption = trekomp(*self.adjustedTimeStampArrayPtr, [k1, k2, k3, k4, vb], plasmaFunc =
*self.arterieInterpolPtr )    ;Interpolated curve
voxelAdaption = trekomp(time interpol, [k1, k2, k3, k4, vb], plasmaFunc =
*self.arterieInterpolPtr )    ;Interpolated curve
correl = (correlate(voxelAdaption, intensity))^2
(*self.pearsonPtr)[xind, yind, zind] = correl
endfor

endif
END

PRO LVA_SliceData::LVA_CalculatePatlakMap, entireFrame
timeStampDimension = size(*(self.timeStampArrayPtr), /dimension)
nrTimestamps = timeStampDimension[1]
timestamp = *self.adjustedTimeStampArrayPtr

plasIntensity = fltarr(nrTimestamps)
for sequence = 0L, self.nrPlasmaPoints -1 do begin
for iter = 0L, nrTimestamps-1 DO BEGIN
plasIntensity[iter] = plasIntensity[iter] + (*self.sliceArrayPtr)[self.plasma[0, sequence],
```

```
self.plasma[1, sequence], self.plasma[2, sequence], iter ]
endfor
endfor

plasIntensity = plasIntensity / self.nrPlasmaPoints *
self.imageIntensityScale*self.weight*1000/(self.injectedActivity/self.injectedActivityConversi
onFactor)

art kum = fltarr(nrTimestamps)
for i=1, nrTimestamps-1 do art_kum[i]= int_tabulated(timestamp(0:i), plasIntensity(0:i),
/DOUBLE)

time_ind=where(timestamp ge 20)
x = art_kum(time_ind)/plasIntensity(time_ind)

for sequence = 0L, self.nrTumorPoints -1 do begin
y = (*self.sliceArrayPtr)[self.tumor[0, sequence], self.tumor[1, sequence], self.tumor[2,
sequence], time_ind ]/plasIntensity(time_ind)*self.SUVConversionFactor
res=linfit(x, y, YFIT=yfit)
(*self.patSlopePtr)[self.tumor[0, sequence], self.tumor[1, sequence], self.tumor[2,
sequence]]=res(1)
(*self.patInterPtr)[self.tumor[0, sequence], self.tumor[1, sequence], self.tumor[2,
sequence]]=res(0)
endfor

if(entireFrame eq 1) then begin
;For all points in current slice
for xind = 0, (size(*self.sliceArrayPtr, /dimensions))[0] -1 do begin
for yind = 0, (size(*self.sliceArrayPtr, /dimensions))[1] -1 do begin
zind = self.depth
y = (*self.sliceArrayPtr)[xind, yind, zind, time_ind
]/plasIntensity(time ind)*self.SUVConversionFactor
if(max(y) gt 0 ) then begin
res=linfit(x, y, YFIT=yfit)
(*self.patSlopePtr)[xind, yind, zind]=res(1)
(*self.patInterPtr)[xind, yind, zind]=res(0)
endif
endfor
endfor

minSlope = min(*self.patSlopePtr)
minInter = min(*self.patInterPtr)

;For all points in current slice
for xind = 0, (size(*self.sliceArrayPtr, /dimensions))[0] -1 do begin
for yind = 0, (size(*self.sliceArrayPtr, /dimensions))[1] -1 do begin
zind = self.depth
y = (*self.sliceArrayPtr)[xind, yind, zind, time_ind
]/plasIntensity(time ind)*self.SUVConversionFactor
if(max(y) gt 0.1 ) then begin
res=linfit(x, y, YFIT=yfit)
(*self.patSlopePtr)[xind, yind, zind]=res(1)
(*self.patInterPtr)[xind, yind, zind]=res(0)
endif else begin
(*self.patSlopePtr)[xind, yind, zind]=minSlope
(*self.patInterPtr)[xind, yind, zind]=minInter
endelse
endfor
endfor
endif

END

FUNCTION LVA_SliceData::LVA_CalculatePatlak
timeStampDimension = size(*(self.timeStampArrayPtr), /dimension)
nrTimestamps = timeStampDimension[1]
timestamp = *self.adjustedTimeStampArrayPtr

plasIntensity = fltarr(nrTimestamps)
for sequence = 0L, self.nrPlasmaPoints -1 do begin
for iter = 0L, nrTimestamps-1 DO BEGIN
plasIntensity[iter] = plasIntensity[iter] + (*self.sliceArrayPtr)[self.plasma[0, sequence],
self.plasma[1, sequence], self.plasma[2, sequence], iter ]
endfor
endfor
```

```
plasIntensity = plasIntensity / self.nrPlasmaPoints *
self.imageIntensityScale*self.weight*1000/(self.injectedActivity/self.injectedActivityConversi
onFactor)

art kum = fltarr(nrTimestamps)
for i=1, nrTimestamps-1 do art_kum[i]= int tabulated(timestamp(0:i), plasIntensity(0:i),
/DOUBLE)

time ind=where(timestamp ge 20)
x = art_kum(time_ind)/plasIntensity(time_ind)

slope = fltarr(self.nrTumorPoints)
intercept = fltarr(self.nrTumorPoints)

for sequence = 0L, self.nrTumorPoints -1 do begin
y = (*self.sliceArrayPtr)[self.tumor[0, sequence], self.tumor[1, sequence], self.tumor[2,
sequence], time_ind ]/plasIntensity(time_ind)*self.SUVConversionFactor
res=linfit(x, y, YFIT=yfit)
slope(sequence)=res(1)
intercept(sequence)=res(0)
endfor

result = [transpose(slope), transpose(intercept)]
return, result
END

;;;;;;;;;;;;;;;;;;;;;;;;
;; Quality calculations ;;
;;;;;;;;;;;;;;;;;;;;;;;;

FUNCTION LVA_SliceData::LVA_CheckKSpace, smoothed
if(N_ELEMENTS(smoothed) eq 0) then smoothed = 0
if( self.nrPlasmaPoints gt 0 AND self.nrTumorPoints gt 0) then begin
;Get image dimensions
imdim = size(*self.sliceArrayPtr, /dimensions)
dim_z = imdim[2]
dim_t = imdim[3]
dim_xy = imdim[0] ; Assuming equal

self->LVA CalculatePlasmaAdaptation
nrTimeInterpolated  = self.interpolFactor* max(*self.adjustedTimeStampArrayPtr)
mx= max(*self.arterieInterpolPtr, location)
time interpol = dindgen(self.interpolFactor*
max(*self.adjustedTimeStampArrayPtr))/self.interpolFactor
timeStamp = double(*self.adjustedTimeStampArrayPtr)

;Create required start points for the 3 compartment function
parinfo = replicate({value:0.0D, fixed:0, limited:[0,0], limits:[0.D,0], relstep: 0.0D}, 5)

parinfo[0].limited = self.k1 limited
parinfo[0].limits  = self.k1_limits
parinfo[1].limited = self.k2_limited
parinfo[1].limits  = self.k2_limits
parinfo[2].limited = self.k3_limited
parinfo[2].limits  = self.k3 limits
parinfo[3].limits = self.k4 limits
parinfo[3].limited = self.k4_limited
parinfo[4].limits  = self.VB_limits
parinfo[4].limited  = self.VB limited
parinfo[0:4].relstep = self.relstep

parinfo[3].fixed = 1

ks = fltarr(4, self.nrTumorPoints, 81)

for run = 0, 80 do begin
k1shift = 1. + (floor(run/27.) -1)/2.
runmod = run - floor(run/27.)*27
k2shift = 1. + (floor(runmod/9.) -1)/2.
runmod = runmod - floor(runmod/9.)*9
k3shift = 1. + (floor(runmod/3.) -1)/2.
stepshift = 1.+ (round(runmod - floor(runmod/3.)*3) -1)/2.
;print,  k1shift, ' ', k2shift, ' ', k3shift, ' ' ,  stepshift
;Initial k estimates
par=double([self.k1_init*k1shift, self.k2_init*k2shift, self.k3_init*k3shift, self.k4_init,
self.VB_init])
```

```
parinfo[0:4].relstep = self.relstep*stepshift
func=dblarr (nrTimeInterpolated-location);(nrTimeInterpolated)
func_err=time_interpol*0+1.0

for tp = 0L, self.nrTumorPoints -1 do begin

x = self.tumor[0,tp]
y = self.tumor[1,tp]
z = self.tumor[2,tp]

if(smoothed eq 1) then begin ;Note that we ignore possible edge problems
fr = 1;
voxelDevelopment = dblarr(1,1,1,dim_t)
voxelCylinder = (*self.sliceArrayPtr)[(x-fr):(x+fr),(y-fr):(y+fr),(z-fr):(z+fr),*]
for j = 0, dim_t-1 do begin
voxelDevelopment(j) = mean( voxelCylinder(*,*,*,j))
endfor
endif else begin
voxelDevelopment = (*self.sliceArrayPtr)[x,y,z,*]
endelse

;Interpolate the slice data to the full sized time array
func_interpol  = double(interpol(voxelDevelopment, timeStamp, time_interpol))
;Calculate
mx= max(*self.arterieInterpolPtr, location)
subTime = time interpol[0:(nrTimeInterpolated-1-location)]
func err=subTime*0+1.0
ttime = subtime;time interpol;[location:*]
tfunc = func_interpol[location:*]
tplas = (*self.arterieInterpolPtr)[location:*]
treKompRes=MPFITFUN('trekomp', ttime, tfunc, func_err, par, PARINFO=parinfo, YFIT=func_fit,
PERROR=perror, STATUS=status, QUIET=1, MAXITER=100, FTOL=1e-6, /DOUBLE, FUNCTARGS =
{plasmaFunc:(tplas)})
;Store
ks[0, tp, run]=treKompRes[0]
ks[1, tp, run]=treKompRes[1]
ks[2, tp, run]=treKompRes[2]
ks[3, tp, run]=treKompRes[4]
endfor
endfor

endif
print, 'Found ', self.nrTumorPoints, ' tumor values'
ksDif = ks[*, *, 1:80] - ks[*, *,0]
k1Mom = moment(ksDif[0,*,*])
k2Mom = moment(ksDif[1,*,*])
k3Mom = moment(ksDif[2,*,*])
vbMom = moment(ksDif[3,*,*])
;stop
return, [max(ksDif[0,*,*]), k1Mom[0], sqrt(k1Mom[1]), max((ksDif[1,*,*])), k2Mom[0],
sqrt(k2Mom[1]),max((ksDif[2,*,*])), k3Mom[0], sqrt(k3Mom[1]), max((ksDif[3,*,*])), vbMom[0],
sqrt(vbMom[1])]
END

FUNCTION LVA_SliceData::LVA_GetPlasmaSUVResiduals
residual = *self.arteriePtr -
(*self.arterieInterpolPtr)[round((*self.adjustedTimeStampArrayPtr)*self.interpolFactor)]
return, residual*self.SUVConversionFactor
END

FUNCTION LVA_SliceData::LVA_GetTumorSUVResiduals
if(ptr_valid(self.timeStampArrayPtr) ne 1 OR ptr_valid(self.k1Ptr) ne 1 ) then return, -1

; To avoid all the indexing we create some temporal variables
timestamp = *self.adjustedTimeStampArrayPtr                    ;Original time values
residuals = fltarr(n_elements(timestamp))

for sequence = 0L, self.nrTumorPoints -1 do begin
k1 = double((*self.k1Ptr)[self.tumor[0, sequence], self.tumor[1, sequence],  self.tumor[2,
sequence]])
k2 = double((*self.k2Ptr)[self.tumor[0, sequence], self.tumor[1, sequence],  self.tumor[2,
sequence]])
k3 = double((*self.k3Ptr)[self.tumor[0, sequence], self.tumor[1, sequence],  self.tumor[2,
sequence]])
k4 = double((*self.k4Ptr)[self.tumor[0, sequence], self.tumor[1, sequence],  self.tumor[2,
sequence]])
```

```
vb = double((*self.vbPtr)[self.tumor[0, sequence], self.tumor[1, sequence],  self.tumor[2,
sequence]])
if( k1 eq 0 AND k2 eq 0 AND k3 eq 0) then begin
print, 'Empty elemetn in tumor'
continue
endif
time_interpol = dindgen(self.interpolFactor*
max(*self.adjustedTimeStampArrayPtr))/self.interpolFactor
intensity = transpose((*self.sliceArrayPtr)[self.tumor[0, sequence], self.tumor[1, sequence],
self.tumor[2, sequence], * ]) ;Original pixel values
voxelAdaption = trekomp(time_interpol, [k1, k2, k3, k4, vb], plasmaFunc =
*self.arterieInterpolPtr )
voxelAdaptionInter = interpol(voxelAdaption, time_interpol, timestamp)
residuals += (intensity -
voxelAdaptionInter)*self.SUVConversionFactor/double(self.nrTumorPoints)
infinitenr = where(finite(residuals) eq 0)
if(infinitenr ne -1 AND n_elements(infinitenr) gt 0) then stop
endfor
return, residuals
END

FUNCTION LVA_SliceData::CheckNoiseResponse
self->LVA_CalculateKSpace, 1
originalK1 = *self.k1Ptr
originalk2 = *self.k2Ptr
originalk3 = *self.k3Ptr
originalVB = *self.VBPtr
nonzero = where(originalk3 ne 0)
originalmb = fltarr(size(originalk3, /dimensions))
originalmb[nonzero] = originalK1[nonzero] *
originalk3[nonzero]/(float(originalk2[nonzero]+originalk3[nonzero]))

;Ruin data
dim t = (size(*self.sliceArrayPtr, /dimensions))[3]
for tind = 0L, dim_t -1 do begin
(*self.sliceArrayPtr)[*, *, *, tind ] = self->LVA_noise((*self.sliceArrayPtr)[*, *, *, tind ],
tind)
endfor

self->LVA CalculateKSpace, 1
newK1 = *self.k1Ptr
newk2 = *self.k2Ptr
newk3 = *self.k3Ptr
newVB = *self.VBPtr
nonzero = where(newk3 ne 0)
newmb = fltarr(size(newk3, /dimensions))
newmb = newK1[nonzero]*newk3[nonzero]/(float(newk2[nonzero]+newk3[nonzero]))

difK1 = (originalK1 - newK1)
difk2 = (originalk2 - newk2)
difk3 = (originalk3 - newk3)
difVB = (originalVB - newVB)
difmb = (originalmb - newmb)

maxk1 = string(max(abs(difK1)), format = '(e10.2)')
meak1 = string((moment(difK1))[0], format = '(e10.2)')
stdk1 = string(sqrt((moment(difK1))[1]), format = '(e10.2)')
maxk2 = string(max(abs(difk2)), format = '(e10.2)')
meak2 = string((moment(difk2))[0], format = '(e10.2)')
stdk2 = string(sqrt((moment(difk2))[1]), format = '(e10.2)')
maxk3 = string(max(abs(difk3)), format = '(e10.2)')
meak3 = string((moment(difk3))[0], format = '(e10.2)')
stdk3 = string(sqrt((moment(difk3))[1]), format = '(e10.2)')
maxvb = string(max(abs(difVB)), format = '(e10.2)')
meavb = string((moment(difVB))[0], format = '(e10.2)')
stdvb = string(sqrt((moment(difVB))[1]), format = '(e10.2)')
maxmb = string(max(abs(difmb)), format = '(e10.2)')
meamb = string((moment(difmb))[0], format = '(e10.2)')
stdmb = string(sqrt((moment(difmb))[1]), format = '(e10.2)')
stop
;return, [maxk1,meak1 ,stdk1, maxk2, meak2, stdk2, maxk3, meak3, stdk3, maxvb, meavb, stdvb]
return, [maxmb,meamb ,stdmb]
END

FUNCTION LVA_SliceData::LVA_Noise, data, time
returnData = data
```

```
if(self.nrTissuePoints gt 0L) then begin
tisArr = fltarr(self.nrTissuePoints)
for sequence = 0L, self.nrTissuePoints -1 do begin
tisArr[sequence] = (*self.sliceArrayPtr)[self.tissue[0, sequence], self.tissue[1, sequence],
self.tissue[2, sequence], time ]
endfor
endif
background = smooth(tisArr, 3)
foreground = tisArr - background
mom = moment(foreground, SDEV = tissueStd)
;stop
;filter
lnoise = randomu(s, size(data, /dimensions))*tissueStd
returnData = data + lnoise
underflow = where(data lt -lnoise)
if(underflow ne -1) then returnData[underflow] = 0
return, returnData
END
```

Display functions

```
PRO LVA_SliceData::LVA_DisplaySlice
if(ptr_valid(self.sliceArrayPtr) eq 1) then begin
xindex = lindgen(self.windowDisplaySize/self.windowZoomFactor)+ self.windowXOffset
yindex = lindgen(self.windowDisplaySize/self.windowZoomFactor)+ self.windowYOffset

if((size(*self.sliceArrayPtr, /dimensions))[0] gt max(xindex) AND (size(*self.sliceArrayPtr,
/dimensions))[1] gt max(yindex)) then begin
sub = ((*self.sliceArrayPtr)[xindex, *, *, *])[*, yindex, self.depth, self.time]
if(self.averagedSUV eq 3 AND self.time lt (size(*self.sliceArrayPtr, /dimensions))[3] -1 AND
self.time gt 0) then begin
sub += ((*self.sliceArrayPtr)[xindex, *, *, *])[*, yindex, self.depth, self.time-1]
sub += ((*self.sliceArrayPtr)[xindex, *, *, *])[*, yindex, self.depth, self.time+1]
sub /= 3.0
endif
if(self.averagedSUV eq 5 AND self.time lt (size(*self.sliceArrayPtr, /dimensions))[3] -1 AND
self.time gt 0) then begin
sub += ((*self.sliceArrayPtr)[xindex, *, *, *])[*, yindex, self.depth, self.time-1]
sub += ((*self.sliceArrayPtr)[xindex, *, *, *])[*, yindex, self.depth, self.time-2]
sub += ((*self.sliceArrayPtr)[xindex, *, *, *])[*, yindex, self.depth, self.time+1]
sub += ((*self.sliceArrayPtr)[xindex, *, *, *])[*, yindex, self.depth, self.time+2]
sub /= 5.0
endif
;SUVNoised = self->LVA_Noise(sub)
SUVMap = sub *
self.imageIntensityScale*self.weight*1000/(self.injectedActivity/self.injectedActivityConversi
onFactor)
SUVExpanded = congrid(SUVMap, self.windowDisplaySize, self.windowDisplaySize)
SUVSmoothed = filter_image( SUVExpanded, SMOOTH=self.windowSmoothSetting,
MEDIAN=self.windowSmoothSetting)
tv, rebin(gmascl(SUVSmoothed, min = self.petDisplayRange[0]/self.dispRngRatio, max =
self.petDisplayRange[1]/self.dispRngRatio, gamma = self.petGamma), self.windowDisplaySize,
self.windowDisplaySize, /sample)
;tv, rebin(imscale(SUVSmoothed, range=[self.displayRange[0],
self.displayRange[1]]/self.dispRngRatio ), self.windowDisplaySize, self.windowDisplaySize,
/sample)
endif else print, 'LVA_DisplaySlice, index out of bounds'

if(self.windowZoomFactor eq 4.0) then begin
plots, self.windowXpos*4.0, self.windowYpos*4.0, LINESTYLE=1, /device
plots, self.windowXpos*4.0+256, self.windowYpos*4.0, LINESTYLE=1, /continue, /device
plots, self.windowXpos*4.0+256, self.windowYpos*4.0+256, LINESTYLE=1 , /continue, /device
plots, self.windowXpos*4.0, self.windowYpos*4.0+256, LINESTYLE=1, /continue, /device
plots, self.windowXpos*4.0, self.windowYpos*4.0, LINESTYLE=1, /continue, /device
endif
endif
END

PRO LVA_SliceData::LVA_DisplayCTSlice
if(ptr_valid(self.ctSliceArrayPtr) eq 1) then begin
CTSubImage = self->LVA_GetCTSubSlice(self.manualCTShift[0], self.manualCTShift[1])

;              equalizedImage = HIST_EQUAL(CTSubImage)
```

```
;              if((size(equalizedImage, /dimensions))[0] eq 192) then begin
;             tv, congrid(imscale(equalizedImage, range=imclip( equalizedImage, PERCENT= 90 )
), self.windowDisplaySize, self.windowDisplaySize)
;         endif else tv, rebin(imscale(equalizedImage, range=imclip( equalizedImage, PERCENT=
90 ) ), self.windowDisplaySize, self.windowDisplaySize, /sample)

;myrange=imclip( CTSubImage, PERCENT= 90 )
;mygamma = 1.7
if((size(CTSubImage, /dimensions))[0] eq 192) then begin
tv, congrid(imscale(CTSubImage, range = self.ctDisplayRange), self.windowDisplaySize,
self.windowDisplaySize)
endif else tv, rebin(imscale(CTSubImage, range = self.ctDisplayRange), self.windowDisplaySize,
self.windowDisplaySize, /sample)
;tv, rebin(gmascl(CTSubImage, min = self.ctDisplayRange[0], max = self.ctDisplayRange[1],
gamma = mygamma), self.windowDisplaySize, self.windowDisplaySize, /sample)
endif
END

FUNCTION LVA_SliceData::LVA_GetCTSubSlice, xshift, yshift
subFrame = 0
if(N_params() lt 2) then begin
xshift = 0
yshift = 0
endif

if(ptr_valid(self.ctSliceArrayPtr) eq 1) then begin
;Image alignment
ctXFOV = (size(*self.ctSliceArrayPtr, /dimension))[0]*self.ctImageResolution(0)
petXFOV = (size(*self.sliceArrayPtr, /dimension))[0]*self.petSliceResolution(0)
ctYFOV = (size(*self.ctSliceArrayPtr, /dimension))[1]*self.ctImageResolution(1)
petYFOV = (size(*self.sliceArrayPtr, /dimension))[1]*self.petSliceResolution(1)
ctZFOV = (size(*self.ctSliceArrayPtr, /dimension))[2]*self.ctImageResolution(2)
petZFOV = (size(*self.sliceArrayPtr, /dimension))[2]*self.petSliceResolution(2)
myDepth = (*self.depthStampArrayPtr)[self.depth]
diff = 1000
depthindex = 0
for i = 0, (size(*self.ctDepthStampArrayPtr, /dimension))[0]-1 do begin
if(abs((*self.ctDepthStampArrayPtr)[i] - myDepth) lt diff) then begin
depthindex = i
diff = abs((*self.ctDepthStampArrayPtr)[i] - myDepth)
endif
endfor

ctXShift = round((self.ctImageCenterPoint[0] -
self.petSlicePosition[0])/(self.ctImageResolution)[0])
ctYShift = round((self.ctImageCenterPoint[1] -
self.petSlicePosition[1])/(self.ctImageResolution)[1])
;ctZShift = self.petSlicePosition(2) - self.ctImageCenterPoint(2)
alignmentShiftedFrame = shift((*self.ctSliceArrayPtr)[*,*,depthindex], ctXShift+xshift,
ctYShift+yshift)
petCTXscaling = (size(*self.ctSliceArrayPtr, /dimension))[0]/(size(*self.sliceArrayPtr,
/dimension))[0]
petCTYscaling = (size(*self.ctSliceArrayPtr, /dimension))[1]/(size(*self.sliceArrayPtr,
/dimension))[1]

xindex = lindgen(petCTXscaling*self.windowDisplaySize/self.windowZoomFactor)+
self.windowXOffset*petCTXscaling
yindex = lindgen(petCTyscaling*self.windowDisplaySize/self.windowZoomFactor)+
self.windowYOffset*petCTYscaling

if((size(*self.ctSliceArrayPtr, /dimensions))[0] gt max(xindex) AND
(size(*self.ctSliceArrayPtr, /dimensions))[1] gt max(yindex)) then begin
subFrame = (alignmentShiftedFrame[xindex, *])[*, yindex]
endif else print, 'LVA_DisplayCTSlice, index out of bounds'

endif
return, subFrame
END

PRO LVA_SliceData::LVA_DisplaySUVBlended
if(ptr_valid(self.sliceArrayPtr) ne 1) then return

xWidth = self.windowDisplaySize
yWidth = self.windowDisplaySize

;Create SUV matrix
```

```
xindex = lindgen(self.windowDisplaySize/self.windowZoomFactor)+ self.windowXOffset
yindex = lindgen(self.windowDisplaySize/self.windowZoomFactor)+ self.windowYOffset

if((size(*self.sliceArrayPtr, /dimensions))[0] gt max(xindex) AND (size(*self.sliceArrayPtr,
/dimensions))[1] gt max(yindex)) then begin
sub = ((*self.sliceArrayPtr)[xindex, *, *, *])[*, yindex, self.depth, self.time]
if(self.averagedSUV eq 3 AND self.time lt (size(*self.sliceArrayPtr, /dimensions))[3] -1 AND
self.time gt 0) then begin
sub += ((*self.sliceArrayPtr)[xindex, *, *, *])[*, yindex, self.depth, self.time-1]
sub += ((*self.sliceArrayPtr)[xindex, *, *, *])[*, yindex, self.depth, self.time+1]
sub /= 3.0
endif
if(self.averagedSUV eq 5 AND self.time lt (size(*self.sliceArrayPtr, /dimensions))[3] -2 AND
self.time gt 1) then begin
sub += ((*self.sliceArrayPtr)[xindex, *, *, *])[*, yindex, self.depth, self.time-1]
sub += ((*self.sliceArrayPtr)[xindex, *, *, *])[*, yindex, self.depth, self.time-2]
sub += ((*self.sliceArrayPtr)[xindex, *, *, *])[*, yindex, self.depth, self.time+1]
sub += ((*self.sliceArrayPtr)[xindex, *, *, *])[*, yindex, self.depth, self.time+2]
sub /= 5.0
endif
SUVMap = sub *
self.imageIntensityScale*self.weight*1000/(self.injectedActivity/self.injectedActivityConversi
onFactor)
endif
;SUVScaled = imscale(SUVMap, range=[self.displayRange[0]/self.dispRngRatio,
self.displayRange[1]]/self.dispRngRatio )
SUVExpand = congrid(SUVMap, xWidth, yWidth)
SUVSmoothed = filter image( SUVExpand, SMOOTH=self.windowSmoothSetting,
MEDIAN=self.windowSmoothSetting)
;SUVGamma = gmascl(SUVSmoothed, min = self.displayRange[0]/self.dispRngRatio, max =
self.displayRange[1]/self.dispRngRatio, gamma = self.petGamma)

;CT image
CTSubImage = self->LVA GetCTSubSlice(self.manualCTShift[0], self.manualCTShift[1])
CTExpand = congrid(CTSubImage, xWidth, yWidth)

CTrange=imclip( CTSubImage, PERCENT= 90 )

;self->LVA Displaybuffers, SUVSmoothed, self.displayRange[0]/self.dispRngRatio,
self.displayRange[1]/self.dispRngRatio, CTExpand, CTrange[0], CTrange[1], 0.5
self->LVA_DisplaybuffersOF, SUVSmoothed, self.petDisplayRange[0]/self.dispRngRatio,
self.petDisplayRange[1]/self.dispRngRatio, CTExpand, self.ctDisplayRange[0],
self.ctDisplayRange[1], 0.4
END

PRO LVA_SliceData::LVA_Displaybuffers, frame1, min1, max1, frame2, min2, max2, alpha
xdim = (size(frame1, /dimension))[0] ;Assuming equal
ydim = (size(frame1, /dimension))[1]
LOADCT, 3, /SILENT
TVLCT, red, green, blue, /GET
image_gate = BYTARR(3, xdim, ydim)
image_gate(0,*,*) = red(bytscl(frame1, min=min1, max=max1))
image_gate(1,*,*) = green(bytscl(frame1, min=min1, max=max1))
image_gate(2,*,*) = blue(bytscl(frame1, min=min1, max=max1))

LOADCT, 0, /SILENT
TVLCT, red, green, blue, /GET
image_CT =BYTARR(3, xdim, ydim)
image_CT(0,*,*) = red(bytscl(frame2, min=min2, max=max2))
image_CT(1,*,*) = green(bytscl(frame2, min=min2, max=max2))
image_CT(2,*,*) = blue(bytscl(frame2, min=min2, max=max2))

image_blend = byte(alpha*float(image_CT)+(1.0-alpha)*float(image_gate))
TV, image_blend, true = 1
END

PRO LVA_SliceData::LVA_DisplaybuffersOF, frame1, min1, max1, frame2, min2, max2, alpha
xdim = (size(frame1, /dimension))[0] ;Assuming equal
ydim = (size(frame1, /dimension))[1]
LOADCT, 3, /SILENT
TVLCT, red, green, blue, /GET
image_gate = BYTARR(3, xdim, ydim)
image_gate(0,*,*) = red(gmascl(frame1, min = min1, max = max1, gamma = self.petGamma))
image_gate(1,*,*) = green(gmascl(frame1, min = min1, max = max1, gamma = self.petGamma))
image_gate(2,*,*) = blue(gmascl(frame1, min = min1, max = max1, gamma = self.petGamma))
```

```
;       image_gate(0,*,*) = red(gmascl(frame1, min = self.displayRange[0]/self.dispRngRatio,
max = self.displayRange[1]/self.dispRngRatio, gamma = self.petGamma))
;       image_gate(1,*,*) = green(gmascl(frame1, min = self.displayRange[0]/self.dispRngRatio,
max = self.displayRange[1]/self.dispRngRatio, gamma = self.petGamma))
;       image_gate(2,*,*) = blue(gmascl(frame1, min = self.displayRange[0]/self.dispRngRatio,
max = self.displayRange[1]/self.dispRngRatio, gamma = self.petGamma))

;       image_gate(0,*,*) = red(bytscl(frame1, min=min1, max=max1))
;       image_gate(1,*,*) = green(bytscl(frame1, min=min1, max=max1))
;       image_gate(2,*,*) = blue(bytscl(frame1, min=min1, max=max1))

LOADCT, 0, /SILENT
TVLCT, red, green, blue, /GET
image_CT =BYTARR(3, xdim, ydim)
image_CT(0,*,*) = red(bytscl(frame2, min=min2, max=max2))
image_CT(1,*,*) = green(bytscl(frame2, min=min2, max=max2))
image_CT(2,*,*) = blue(bytscl(frame2, min=min2, max=max2))

maxVal = 255
addframe = (alpha)*image_gate
overflow = where(replicate(maxVal, xdim, ydim, 3)-addframe lt image_CT)
blendframe = image_CT + addframe
if((size(overflow, /dimensions))[0] gt 0) then blendframe[overflow] = maxVal
TV, blendframe, true = 1
END

PRO LVA_SliceData::LVA_DisplayKSlice, kDimension
xindex = lindgen(self.windowDisplaySize/self.windowZoomFactor)+ self.windowXOffset
yindex = lindgen(self.windowDisplaySize/self.windowZoomFactor)+ self.windowYOffset

;Assuming dimensions equal
if(ptr_valid(self.k1Ptr) eq 1 AND ptr_valid(self.k2Ptr) eq 1 AND ptr_valid(self.k3Ptr) eq 1)
then begin
if((size(*self.k1Ptr, /dimensions))[0] gt max(xindex) AND (size(*self.k1Ptr, /dimensions))[1]
gt max(yindex)) then begin
;range=[self.displayRange[0], self.displayRange[1]] )
if(kDimension eq 1) then sub = 300.0d*((*self.k1Ptr)[xindex, *, *])[*, yindex, self.depth]
if(kDimension eq 2) then sub = 300.0d*((*self.k2Ptr)[xindex, *, *])[*, yindex, self.depth]
if(kDimension eq 3) then sub = 300.0d*((*self.k3Ptr)[xindex, *, *])[*, yindex, self.depth]
if(kDimension eq 4) then sub = 300.0d*((*self.k4Ptr)[xindex, *, *])[*, yindex, self.depth]
if(max(sub) gt 0) then tv, rebin(imscale(sub, range=imclip( sub, PERCENT= 90 ) ),
self.windowDisplaySize, self.windowDisplaySize, /sample)
endif else print, 'LVA_DisplayKSlice, index out of bounds'
endif
END

PRO LVA_SliceData::LVA_DisplayKComposite, knr
case knr of
1 : begin
kPtr = self.k1Ptr
kDispRngRat = 30.
ldisprng = self.K1DisplayRange[0]/self.dispRngRatio*kDispRngRat
hdisprng = self.K1DisplayRange[1]/self.dispRngRatio*kDispRngRat
endcase
2 : begin
kPtr = self.k2Ptr
kDispRngRat = 50.
ldisprng = self.k2DisplayRange[0]/self.dispRngRatio*kDispRngRat
hdisprng = self.k2DisplayRange[1]/self.dispRngRatio*kDispRngRat
endcase
3 : begin
kptr = self.k3Ptr
kDispRngRat = 2.5
ldisprng = self.k3DisplayRange[0]/self.dispRngRatio*kDispRngRat
hdisprng = self.k3DisplayRange[1]/self.dispRngRatio*kDispRngRat
endcase
endcase
if(ptr_valid(kPtr) ne 1) then return

xWidth = self.windowDisplaySize
yWidth = self.windowDisplaySize

;Create k matrix
xindex = lindgen(self.windowDisplaySize/self.windowZoomFactor)+ self.windowXOffset
yindex = lindgen(self.windowDisplaySize/self.windowZoomFactor)+ self.windowYOffset
```

```
if((size(*kPtr, /dimensions))[0] gt max(xindex) AND (size(*kPtr, /dimensions))[1] gt
max(yindex)) then begin
sub = 300.0d*((*kPtr)[xindex, *, *])[*, yindex, self.depth]
endif

if(max(sub) gt 0) then begin
KExpand = congrid(sub, xWidth, yWidth)
KSmoothed = filter_image( KExpand, SMOOTH=self.windowSmoothSetting,
MEDIAN=self.windowSmoothSetting)

;CT image
CTSubImage = self->LVA_GetCTSubSlice(self.manualCTShift[0], self.manualCTShift[1])
CTExpand = congrid(CTSubImage, xWidth, yWidth)

self->LVA DisplaybuffersOF, KSmoothed, ldisprng, hdisprng, CTExpand, self.ctDisplayRange[0],
self.ctDisplayRange[1], 0.4
endif
END

PRO LVA_SliceData::LVA_DisplayMetabolicComposite
if(ptr_valid(self.k1Ptr) ne 1) then return
kDispRngRat = 1.0
corrupt = 0
xWidth = self.windowDisplaySize
yWidth = self.windowDisplaySize

;Create k matrix
xindex = lindgen(self.windowDisplaySize/self.windowZoomFactor)+ self.windowXOffset
yindex = lindgen(self.windowDisplaySize/self.windowZoomFactor)+ self.windowYOffset

if((size(*self.k1Ptr, /dimensions))[0] gt max(xindex) AND (size(*self.k1Ptr, /dimensions))[1]
gt max(yindex)) then begin
sub = 300.0d*((*self.k1Ptr)[xindex, *, *])[*, yindex, self.depth]*((*self.k3Ptr)[xindex, *,
*])[*, yindex, self.depth]
subk2 = ((*self.k2Ptr)[xindex, *, *])[*, yindex, self.depth]
subk3 = ((*self.k3Ptr)[xindex, *, *])[*, yindex, self.depth]
nonzeroIndex = where(subk2 gt 0.0)
if(max(nonzeroIndex) gt -1) then sub[nonzeroIndex] /=
(subk2[nonzeroIndex]+subk3[nonzeroIndex]) else corrupt = 1
endif

if(corrupt eq 0) then begin
if(max(sub) gt 0) then begin
KExpand = congrid(sub, xWidth, yWidth)
KSmoothed = filter_image( KExpand, SMOOTH=self.windowSmoothSetting,
MEDIAN=self.windowSmoothSetting)

;CT image
CTSubImage = self->LVA GetCTSubSlice(self.manualCTShift[0], self.manualCTShift[1])
CTExpand = congrid(CTSubImage, xWidth, yWidth)

self->LVA_DisplaybuffersOF, KSmoothed, self.mbDisplayRange[0]/self.dispRngRatio*kDispRngRat,
self.mbDisplayRange[1]/self.dispRngRatio*kDispRngRat, CTExpand, self.ctDisplayRange[0],
self.ctDisplayRange[1], 0.4
endif
endif else begin
clearbuff = replicate(0, 512, 512)
tv, clearbuff
endelse
END

PRO LVA_SliceData::LVA_DisplayVBSlice
xindex = lindgen(self.windowDisplaySize/self.windowZoomFactor)+ self.windowXOffset
yindex = lindgen(self.windowDisplaySize/self.windowZoomFactor)+ self.windowYOffset

if(ptr_valid(self.VBPtr) eq 1) then begin
if((size(*self.VBPtr, /dimensions))[0] gt max(xindex) AND (size(*self.VBPtr, /dimensions))[1]
gt max(yindex)) then begin
;range=[self.displayRange[0], self.displayRange[1]] )
sub = 300.0d*((*self.VBPtr)[xindex, *, *])[*, yindex, self.depth]
if(max(sub) gt 0) then tv, rebin(imscale(sub, range=imclip( sub, PERCENT= 90 ) ),
self.windowDisplaySize, self.windowDisplaySize, /sample)
endif else print, 'LVA_DisplayVBSlice, index out of bounds'
endif
END
```

```
PRO LVA_SliceData::LVA_DisplayVBComposite

vbDispRngRat = 15.
if(ptr_valid(self.VBPtr) ne 1) then return

xWidth = self.windowDisplaySize
yWidth = self.windowDisplaySize

;Create VB matrix
xindex = lindgen(self.windowDisplaySize/self.windowZoomFactor)+ self.windowXOffset
yindex = lindgen(self.windowDisplaySize/self.windowZoomFactor)+ self.windowYOffset

if((size(*self.VBPtr, /dimensions))[0] gt max(xindex) AND (size(*self.VBPtr, /dimensions))[1]
gt max(yindex)) then begin
sub = 300.0d*((*self.VBPtr)[xindex, *, *])[*, yindex, self.depth]
endif

if(max(sub) gt 0) then begin
KExpand = congrid(sub, xWidth, yWidth)
KSmoothed = filter image( KExpand, SMOOTH=self.windowSmoothSetting,
MEDIAN=self.windowSmoothSetting)

;CT image
CTSubImage = self->LVA_GetCTSubSlice(self.manualCTShift[0], self.manualCTShift[1])
CTExpand = congrid(CTSubImage, xWidth, yWidth)

self->LVA DisplaybuffersOF, KSmoothed, self.vbDisplayRange[0]/self.dispRngRatio*vbDispRngRat,
self.vbDisplayRange[1]/self.dispRngRatio*vbDispRngRat, CTExpand, self.ctDisplayRange[0],
self.ctDisplayRange[1], 0.4
endif
END

PRO LVA_SliceData::LVA_DisplayR2Slice
xindex = lindgen(self.windowDisplaySize/self.windowZoomFactor)+ self.windowXOffset
yindex = lindgen(self.windowDisplaySize/self.windowZoomFactor)+ self.windowYOffset

;Assuming dimensions equal
if(ptr_valid(self.pearsonPtr) eq 1 ) then begin
if((size(*self.pearsonPtr, /dimensions))[0] gt max(xindex) AND (size(*self.pearsonPtr,
/dimensions))[1] gt max(yindex)) then begin
sub = 255.0d*((*self.pearsonPtr)[xindex, *, *])[*, yindex, self.depth]
if(max(sub) gt 0) then tv, rebin(imscale(sub, range=imclip( sub, PERCENT= 90 ) ),
self.windowDisplaySize, self.windowDisplaySize, /sample)
endif else print, 'LVA_DisplayR2Slice, index out of bounds'
endif
END

PRO LVA_SliceData::LVA_DisplayPatlakSlopeSlice
xindex = lindgen(self.windowDisplaySize/self.windowZoomFactor)+ self.windowXOffset
yindex = lindgen(self.windowDisplaySize/self.windowZoomFactor)+ self.windowYOffset

xWidth = self.windowDisplaySize
yWidth = self.windowDisplaySize
;Assuming dimensions equal
if(ptr_valid(self.patSlopePtr) eq 1 ) then begin
if((size(*self.patSlopePtr, /dimensions))[0] gt max(xindex) AND (size(*self.patSlopePtr,
/dimensions))[1] gt max(yindex)) then begin
sub = 100.0d*((*self.patSlopePtr)[xindex, *, *])[*, yindex, self.depth]
if(max(sub) gt 0) then begin

maxSlop = 0
minSlop = 1000
for sequence = 0L, self.nrTumorPoints -1 do begin
myval = (*self.patSlopePtr)[self.tumor[0, sequence], self.tumor[1, sequence], self.tumor[2,
sequence]]
if(myval gt maxSlop) then maxSlop = myval
if(myval lt minSlop) then minSlop = myval
endfor

PatExpand = congrid(sub, xWidth, yWidth)
PatSmoothed = filter image( PatExpand, SMOOTH=self.windowSmoothSetting,
MEDIAN=self.windowSmoothSetting)

patrange=imclip( PatSmoothed, PERCENT= 80 )
patrange[0] += (patrange[1] - patrange[0])*0.25
```

```
if(minSlop gt patrange[0]) then patrange[0] = (1.5*minSlop+patrange[0])/2
if(maxSlop lt patrange[1]) then patrange[1] = (1.5*maxSlop+patrange[1])/3

;CT image
CTSubImage = self->LVA GetCTSubSlice(self.manualCTShift[0], self.manualCTShift[1])
CTExpand = congrid(CTSubImage, xWidth, yWidth)

CTrange=imclip( CTSubImage, PERCENT= 90 )

self->LVA DisplaybuffersOF, PatSmoothed, self.psDisplayRange[0]/self.dispRngRatio,
self.psDisplayRange[1]/self.dispRngRatio, CTExpand, self.ctDisplayRange[0],
self.ctDisplayRange[1], 0.4
endif
endif else print, 'LVA_PatlakSlopeSlice, index out of bounds'
endif
END

PRO LVA_SliceData::LVA_DisplayPatlakInterceptSlice
xindex = lindgen(self.windowDisplaySize/self.windowZoomFactor)+ self.windowXOffset
yindex = lindgen(self.windowDisplaySize/self.windowZoomFactor)+ self.windowYOffset

xWidth = self.windowDisplaySize
yWidth = self.windowDisplaySize
;Assuming dimensions equal
if(ptr_valid(self.patInterPtr) eq 1 ) then begin
if((size(*self.patInterPtr, /dimensions))[0] gt max(xindex) AND (size(*self.patInterPtr,
/dimensions))[1] gt max(yindex)) then begin
sub = 100.0d*((*self.patInterPtr)[xindex, *, *])[*, yindex, self.depth]
if(max(sub) gt 0) then begin
PatExpand = congrid(sub, xWidth, yWidth)
PatSmoothed = filter_image( PatExpand, SMOOTH=self.windowSmoothSetting,
MEDIAN=self.windowSmoothSetting)

patrange=imclip( PatSmoothed, PERCENT= 80 )
patrange[0] += (patrange[1] - patrange[0])*0.15
;CT image
CTSubImage = self->LVA_GetCTSubSlice(self.manualCTShift[0], self.manualCTShift[1])
CTExpand = congrid(CTSubImage, xWidth, yWidth)

CTrange=imclip( CTSubImage, PERCENT= 90 )

self->LVA_DisplaybuffersOF, PatSmoothed, self.piDisplayRange[0]/self.dispRngRatio*10.,
self.piDisplayRange[1]/self.dispRngRatio*10., CTExpand, self.ctDisplayRange[0],
self.ctDisplayRange[1], 0.4
endif
endif else print, 'LVA_PatlakSlopeSlice, index out of bounds'
endif
END

PRO LVA_SliceData::LVA_DisplayTumorWindow
if(self.displayTumorWindow eq 0 OR self.windowTumorXsize eq 0 OR self.windowTumorYsize eq 0)
then return

zf = self.windowZoomFactor/8.0
plots, self.windowTumorXpos*zf                          , self.windowTumorYpos*zf
, LINESTYLE=1, /device
plots, self.windowTumorXpos*zf+self.windowTumorXsize*zf , self.windowTumorYpos*zf
, LINESTYLE=1, /continue, /device
plots, self.windowTumorXpos*zf+self.windowTumorXsize*zf ,
self.windowTumorYpos*zf+self.windowTumorYsize*zf  , LINESTYLE=1, /continue, /device
plots, self.windowTumorXpos*zf                          ,
self.windowTumorYpos*zf+self.windowTumorYsize*zf  , LINESTYLE=1, /continue, /device
plots, self.windowTumorXpos*zf                          , self.windowTumorYpos*zf
, LINESTYLE=1, /continue, /device
END

PRO LVA_SliceData::LVA_DisplayOutputWindow
if(self.displayOutputWindow eq 0 OR self.windowOutputXsize eq 0 OR self.windowOutputYsize eq
0) then return

zf = self.windowZoomFactor/8.0

plots, self.windowOutputXpos*zf                            , self.windowOutputYpos*zf
, LINESTYLE=1, /device
plots, self.windowOutputXpos*zf+self.windowOutputXsize*zf  , self.windowOutputYpos*zf
, LINESTYLE=1, /continue, /device
```

```
plots, self.windowOutputXpos*zf+self.windowOutputXsize*zf   ,
self.windowOutputYpos*zf+self.windowOutputYsize*zf, LINESTYLE=1, /continue, /device
plots, self.windowOutputXpos*zf                             ,
self.windowOutputYpos*zf+self.windowOutputYsize*zf, LINESTYLE=1, /continue, /device
plots, self.windowOutputXpos*zf                             , self.windowOutputYpos*zf
, LINESTYLE=1, /continue, /device
END

;Draw the boxes representing plasma pixels
PRO LVA_SliceData::LVA_DisplayPlasma
if(self.displayPlasmaROI eq 1) then begin
self->LVA_CalculatePlasmaFunction
for pindex = 0, self.nrPlasmaPoints-1 do begin
if(self.plasma[2, pindex] eq self.depth) then begin
;Adding a slight boundary
x1 = (self.plasma[0, pindex] - self.windowXOffset )*self.windowZoomFactor +
self.windowZoomFactor/4.0
;Drawing a halfsize pixel
x2 = x1 + self.windowZoomFactor/2.0
y1 = (self.plasma[1, pindex] - self.windowYOffset)*self.windowZoomFactor -1
+self.windowZoomFactor/4.0
y2 = y1 + self.windowZoomFactor/2.0
x = [x1, x2, x2, x1]
y = [y1, y1, y2, y2]
POLYFILL, x, y, color = 175, /DEVICE ; (175, 0 , )
endif
endfor
endif
END

;Draw the boxes representing tumor pixels
PRO LVA_SliceData::LVA_DisplayTumor
if(self.displayTumorROI eq 1) then begin
for pindex = 0, self.nrTumorPoints-1 do begin
if(self.tumor[2, pindex] eq self.depth) then begin
x1 = (self.tumor[0, pindex]- self.windowXOffset)*self.windowZoomFactor +
self.windowZoomFactor/4.0
x2 = x1 + self.windowZoomFactor/2.0
y1 = (self.tumor[1, pindex]- self.windowYOffset)*self.windowZoomFactor -1
+self.windowZoomFactor/4.0
y2 = y1 + self.windowZoomFactor/2.0
x = [x1, x2, x2, x1]
y = [y1, y1, y2, y2]
POLYFILL, x, y, color = 32768, /DEVICE; (128, 128, 0)
endif
endfor
endif
END

;Draw the tumor pixels
PRO LVA_SliceData::LVA_DisplayContinousTumor
if(self.displayTumorROI eq 1) then begin
for pindex = 0, self.nrTumorPoints-1 do begin
if(self.tumor[2, pindex] eq self.depth) then begin
x1 = (self.tumor[0, pindex]- self.windowXOffset)*self.windowZoomFactor
x2 = x1 + self.windowZoomFactor
y1 = (self.tumor[1, pindex]- self.windowYOffset)*self.windowZoomFactor -1
y2 = y1 + self.windowZoomFactor
x = [x1, x2, x2, x1]
y = [y1, y1, y2, y2]
POLYFILL, x, y, color = 32768, /DEVICE; (128, 128, 0)
endif
endfor
endif
END

PRO LVA_SliceData::LVA_BlendTumorOntoFrame, smoothing, activeWindow
baseFrame = TVREAD(TRUE=3)

Window, XSIZE = 512, YSIZE = 512, /FREE, /PIXMAP
pixmapWin = !D.Window
WSet, pixmapWin
self->LVA_DisplayContinousTumor
tumorFrame = TVREAD(TRUE=3)
WDelete, pixmapWin
wset, activeWindow
```

```
alpha = 0.3
maxVal = max(baseFrame)
if(size(size(tumorFrame, /dimensions), /dimensions) eq 3) then begin
redSmoothed = filter image( tumorFrame[*,*,0], SMOOTH=smoothing, MEDIAN=smoothing)
greenSmoothed = filter image( tumorFrame[*,*,1], SMOOTH=smoothing, MEDIAN=smoothing)
blueSmoothed = filter image( tumorFrame[*,*,2], SMOOTH=smoothing, MEDIAN=smoothing)
tumorFrame[*,*,0] = redSmoothed
tumorFrame[*,*,1] = greenSmoothed
tumorFrame[*,*,2] = blueSmoothed
addframe = (1-alpha)*tumorFrame
overflow = where(replicate(maxVal, 512, 512, 3)-addframe lt baseFrame)
blendframe = baseFrame + addframe
if((size(overflow, /dimensions))[0] gt 0) then blendframe[overflow] = maxVal
TV, blendframe, TRUE=3
endif else begin
tumorFrame = filter image( tumorFrame, SMOOTH=smoothing, MEDIAN=smoothing)
addframe = (1-alpha)*tumorFrame
overflow = where(replicate(maxVal, 512, 512)-addframe lt baseFrame)
blendframe = baseFrame + addframe
if((size(overflow, /dimensions))[0] gt 0) then blendframe[overflow] = maxVal
TV, blendframe
endelse
END

PRO LVA_SliceData::LVA_BlendEdgeTumorOntoFrame, smoothing, activeWindow
baseFrame = TVREAD(TRUE=3)

Window, XSIZE = 512, YSIZE = 512, /FREE, /PIXMAP
pixmapWin = !D.Window
WSet, pixmapWin
self->LVA_DisplayContinousTumor
tumorFrame = TVREAD(TRUE=3)

WDelete, pixmapWin
wset, activeWindow
alpha = 0.3
maxVal = max(baseFrame)
if(size(size(tumorFrame, /dimensions), /dimensions) eq 3) then begin
redSmoothed = filter image( tumorFrame[*,*,0], SMOOTH=smoothing, MEDIAN=smoothing)
greenSmoothed = filter image( tumorFrame[*,*,1], SMOOTH=smoothing, MEDIAN=smoothing)
blueSmoothed = filter_image( tumorFrame[*,*,2], SMOOTH=smoothing, MEDIAN=smoothing)

tumorFrame[*,*,0] = greenSmoothed
tumorFrame[*,*,1] = greenSmoothed
tumorFrame[*,*,2] = greenSmoothed

radius = 2
strucElem = SHIFT(DIST(2*radius+1), radius, radius) LE radius
growth = DILATE(greenSmoothed/80, strucElem)
edge = growth - greenSmoothed/80

tumorFrame[*,*,0] = redSmoothed
tumorFrame[*,*,1] = edge*255
tumorFrame[*,*,2] = blueSmoothed
addframe = (1-alpha)*tumorFrame
overflow = where(replicate(maxVal, 512, 512, 3)-addframe lt baseFrame)
blendframe = baseFrame + addframe
if((size(overflow, /dimensions))[0] gt 0) then blendframe[overflow] = maxVal
TV, blendframe, TRUE=3
endif else begin
tumorFrame = filter_image( tumorFrame, SMOOTH=smoothing, MEDIAN=smoothing)
radius = 2
strucElem = SHIFT(DIST(2*radius+1), radius, radius) LE radius
morphImg = DILATE(tumorFrame/80, strucElem)
edge = growth - tumorFrame/80
addframe = (1-alpha)*morphImg*255
overflow = where(replicate(maxVal, 512, 512)-addframe lt baseFrame)
blendframe = baseFrame + addframe
if((size(overflow, /dimensions))[0] gt 0) then blendframe[overflow] = maxVal
TV, blendframe
endelse
END

;Draw the boxes representing tissue pixels
PRO LVA_SliceData::LVA_DisplayTissue
if(self.displayTissueROI eq 1) then begin
```

```
for pindex = 0, self.nrTissuePoints-1 do begin
if(self.tissue[2, pindex] eq self.depth) then begin
;Adding a slight boundary
x1 = (self.tissue[0, pindex] - self.windowXOffset )*self.windowZoomFactor +
self.windowZoomFactor/4.0
;Drawing a halfsize pixel
x2 = x1 + self.windowZoomFactor/2.0
y1 = (self.tissue[1, pindex] - self.windowYOffset)*self.windowZoomFactor -1
+self.windowZoomFactor/4.0
y2 = y1 + self.windowZoomFactor/2.0
x = [x1, x2, x2, x1]
y = [y1, y1, y2, y2]
POLYFILL, x, y, color = 8398608, /DEVICE ; (0, 0 , 128)
endif
endfor
endif
END


PRO LVA_SliceData::LVA_DisplaySelectedPixel
if((self.selectedX lt 0 AND self.selectedY lt 0) eq 0 AND self.selectedZ eq self.depth) then
begin
x1 = (self.selectedX - self.windowXOffset)*self.windowZoomFactor + self.windowZoomFactor/8.0
y1 = (self.selectedY - self.windowYOffset)*self.windowZoomFactor
x2 = x1 +self.windowZoomFactor*6.0/8.0
y2 = y1 +self.windowZoomFactor*6.0/8.0
x = [x1, x2, x2, x1]
y = [y1, y1, y2, y2]
POLYFILL, x, y, color = 16581375, /DEVICE; (0, 0, 255)
endif
END

PRO LVA_SliceData::LVA_DisplayHistogram
;Our histogram
if(ptr_valid(self.sliceArrayPtr) eq 1) then begin
sub = (*self.sliceArrayPtr)[*,*, self.depth, self.time]
SUVMap = sub *
self.imageIntensityScale*self.weight*1000/(self.injectedActivity/self.injectedActivityConversi
onFactor)
if(max(SUVMap) gt 0) then begin
hindex = where(SUVMap gt self.petDisplayRange[0]/self.dispRngRatio)
if( size(hindex, /dimensions) gt 0) then begin
histRange = imclip(SUVMap[hindex], PERCENT= 100)
hist plot, SUVMap[hindex], /normal, /fill, xstyle=3
;oplot, [histRange[0], histRange[0]], [0.0, 1.0], linestyle=2
oplot, [histRange[1], histRange[1]], [0.0, 1.0], linestyle=2
endif else print, 'No values greater than ', self.petDisplayRange[0]/self.dispRngRatio
endif
endif
END

PRO LVA_SliceData::LVA_DisplayCTHistogram
;Our histogram
if(ptr_valid(self.ctSliceArrayPtr) eq 1) then begin
sub = (*self.ctSliceArrayPtr)[*,*]
if(max(sub) gt 0) then begin
hindex = where(sub gt max(sub)*self.ctDisplayRange[0]/1000.)
if((size(hindex))[1] gt 0) then begin
histRange = imclip(sub[hindex], PERCENT= 100)
hist plot, sub[hindex], /normal, /fill, xstyle=3
;oplot, [histRange[0], histRange[0]], [0.0, 1.0], linestyle=2
oplot, [histRange[1], histRange[1]], [0.0, 1.0], linestyle=2
endif
endif
endif
END

PRO LVA_SliceData::LVA_DisplayKHistogram
;Our histogram
if(ptr_valid(self.k1Ptr) eq 1) then begin
sub = (*self.k1Ptr)[*,*, self.depth]
if(max(sub) gt 0) then begin
hindex = where(sub gt max(sub)*self.k1DisplayRange[0]/1000.)
if((size(hindex))[1] gt 0) then begin
histRange = imclip(sub[hindex], PERCENT= 100)
hist_plot, sub[hindex], /normal, /fill, xstyle=3
```

```
;oplot, [histRange[0], histRange[0]], [0.0, 1.0], linestyle=2
oplot, [histRange[1], histRange[1]], [0.0, 1.0], linestyle=2
endif
endif
endif
END

;Plot the plasma function over time
PRO LVA_SliceData::LVA_PlotPlasma
if(ptr_valid(self.timeStampArrayPtr) eq 1 AND self.nrPlasmaPoints gt 0L) then begin
originalPlasmaSUVMap =
(*self.arteriePtr)*self.imageIntensityScale*self.weight*1000.0/(self.injectedActivity/float(se
lf.injectedActivityConversionFactor))
plot, *self.adjustedTimeStampArrayPtr, originalPlasmaSUVMap, /YSTYLE, psym=4, color =
self.foregroundColor, background = self.backgroundColor, xtitle = 'Time(min)', ytitle = 'SUV',
charsize = 2
if(ptr_valid(self.arterieInterpolPtr) eq 1) then begin
;Interpolate the time axis
time_interpol = dindgen(self.interpolFactor*
max(*self.adjustedTimeStampArrayPtr))/self.interpolFactor
;intensity interpol  = double(interpol(*self.arteriePtr, *self.adjustedTimeStampArrayPtr,
time interpol))
interpolatedPlasmaSUVMap =
(*self.arterieInterpolPtr)*self.imageIntensityScale*self.weight*1000.0/(self.injectedActivity/
float(self.injectedActivityConversionFactor))

oplot, time interpol, interpolatedPlasmaSUVMap, psym=-3, color = self.foregroundColor
correl = (correlate(*self.arterieInterpolPtr, *self.arteriePtr))^2
xyouts, 350, 220, 'R2:     ' + strtrim(string(correl, format='(F6.2)'), 1), /DEVICE, color =
self.foregroundColor
xyouts, 350, 200, 'Points: ' + strtrim(string(self.validPlasmaPoints), 1), /DEVICE, color =
self.foregroundColor
endif
endif
END

;Plot the tissue variation over time
PRO LVA SliceData::LVA PlotTissue
if(ptr valid(self.timeStampArrayPtr) eq 1 AND self.nrTissuePoints gt 0L) then begin
timeStampDimension = size(*(self.timeStampArrayPtr), /dimension)
nrTimestamps = timeStampDimension[1]
intensity = fltarr(nrTimestamps)
timestamp = *self.adjustedTimeStampArrayPtr
for sequence = 0L, self.nrTissuePoints -1 do begin
for iter = 0L, nrTimestamps-1 DO BEGIN
intensity[iter] = intensity[iter] + (*self.sliceArrayPtr)[self.tissue[0, sequence],
self.tissue[1, sequence], self.tissue[2, sequence], iter ]
endfor
endfor

if(self.nrTissuePoints gt 0) then intensity = intensity / self.nrTissuePoints else intensity =
0
tissueSuv = intensity *
self.imageIntensityScale*self.weight*1000/(self.injectedActivity/self.injectedActivityConversi
onFactor)
plot, timeStamp, tissueSuv, /YSTYLE, psym=4
endif
END

;Plot the tumor variation over time
PRO LVA SliceData::LVA PlotTumor
if(ptr valid(self.timeStampArrayPtr) eq 1 AND self.nrTumorPoints gt 0L) then begin
timeStampDimension = size(*(self.timeStampArrayPtr), /dimension)
nrTimestamps = timeStampDimension[1]
intensity = fltarr(nrTimestamps)
timestamp = *self.adjustedTimeStampArrayPtr
;timeStamp = fltarr(nrTimestamps)
;startTime = (*self.timeStampArrayPtr)[0,0]
;timeStamp = (*self.timeStampArrayPtr)[0,*] - startTime
for sequence = 0L, self.nrTumorPoints -1 do begin
for iter = 0L, nrTimestamps-1 DO BEGIN
intensity[iter] = intensity[iter] + (*self.sliceArrayPtr)[self.tumor[0, sequence],
self.tumor[1, sequence], self.tumor[2, sequence], iter ]
endfor
endfor
```

```
if(self.nrTumorPoints gt 0) then intensity = intensity / self.nrTumorPoints else intensity = 0

tumorSuv = intensity *
self.imageIntensityScale*self.weight*1000/(self.injectedActivity/self.injectedActivityConversi
onFactor)
plot, timeStamp, tumorSuv, /YSTYLE, psym=4
endif
END

PRO LVA SliceData::LVA PlotTumorTissueRatio
if(ptr valid(self.timeStampArrayPtr) eq 1 AND self.nrTumorPoints gt 0L AND self.nrTissuePoints
gt 0L) then begin
timeStampDimension = size(*(self.timeStampArrayPtr), /dimension)
nrTimestamps = timeStampDimension[1]
timestamp = *self.adjustedTimeStampArrayPtr

tumIntensity = fltarr(nrTimestamps)
for sequence = 0L, self.nrTumorPoints -1 do begin
for iter = 0L, nrTimestamps-1 DO BEGIN
tumIntensity[iter] = tumIntensity[iter] + (*self.sliceArrayPtr)[self.tumor[0, sequence],
self.tumor[1, sequence], self.tumor[2, sequence], iter ]
endfor
endfor

tisIntensity = fltarr(nrTimestamps)
for sequence = 0L, self.nrTissuePoints -1 do begin
for iter = 0L, nrTimestamps-1 DO BEGIN
tisIntensity[iter] = tisIntensity[iter] + (*self.sliceArrayPtr)[self.tissue[0, sequence],
self.tissue[1, sequence], self.tissue[2, sequence], iter ]
endfor
endfor

tumIntensity = tumIntensity / self.nrTumorPoints
tisIntensity = tisIntensity / self.nrTissuePoints
smTum = smooth(tumIntensity, 3)
smTis = smooth(tisIntensity, 3)
ratio = smTum / smTis

plot, timeStamp, ratio, /YSTYLE, psym=4
endif
END

PRO LVA SliceData::LVA PlotTumorPlasmaRatio
if(ptr valid(self.timeStampArrayPtr) eq 1 AND self.nrTumorPoints gt 0L AND self.nrPlasmaPoints
gt 0L) then begin
timeStampDimension = size(*(self.timeStampArrayPtr), /dimension)
nrTimestamps = timeStampDimension[1]
timestamp = *self.adjustedTimeStampArrayPtr

tumIntensity = fltarr(nrTimestamps)
for sequence = 0L, self.nrTumorPoints -1 do begin
for iter = 0L, nrTimestamps-1 DO BEGIN
tumIntensity[iter] = tumIntensity[iter] + (*self.sliceArrayPtr)[self.tumor[0, sequence],
self.tumor[1, sequence], self.tumor[2, sequence], iter ]
endfor
endfor

plasIntensity = fltarr(nrTimestamps)
for sequence = 0L, self.nrPlasmaPoints -1 do begin
for iter = 0L, nrTimestamps-1 DO BEGIN
plasIntensity[iter] = plasIntensity[iter] + (*self.sliceArrayPtr)[self.plasma[0, sequence],
self.plasma[1, sequence], self.plasma[2, sequence], iter ]
endfor
endfor

tumIntensity = tumIntensity / self.nrTumorPoints
plasIntensity = plasIntensity / self.nrPlasmaPoints

nrInterpolated = self.interpolFactor* max(*self.adjustedTimeStampArrayPtr)
time interpol = dindgen(nrInterpolated)/self.interpolFactor
tumIntensity interpol  = double(interpol(tumIntensity, timeStamp, time interpol))
plaIntensity_interpol  = double(interpol(plasIntensity, timeStamp, time_interpol))+1e-6

kvota = tumIntensity_interpol/plaIntensity_interpol
timeproduct = plaIntensity_interpol*time_interpol
```

```
rintegral = fltarr(nrInterpolated)
;rintegral = total(timeproduct, /CUMULATIVE)/plaIntensity_interpol
for index = 0L, nrInterpolated-1 do begin
rintegral[index] = total(timeproduct[0:index])/plaIntensity_interpol[index]
endfor

;res = self->LVA_CalculatePatlak()
;plot, res[0,*], res[1,*], /YSTYLE, psym=4


plasIntensity = plasIntensity ;*
self.imageIntensityScale*self.weight*1000/(self.injectedActivity/self.injectedActivityConversi
onFactor)

art kum = fltarr(nrTimestamps)
for i=1, nrTimestamps-1 do art_kum[i]= int_tabulated(timestamp(0:i), plasIntensity(0:i),
/DOUBLE)

time_ind=where(timestamp ge 20)
x = art kum(time ind)/plasIntensity(time ind)
y = tumIntensity(time ind)/plasIntensity(time ind);*
self.imageIntensityScale*self.weight*1000/(self.injectedActivity/self.injectedActivityConversi
onFactor)

fullx = art_kum/plasIntensity
fully = tumIntensity/plasIntensity;*
self.imageIntensityScale*self.weight*1000/(self.injectedActivity/self.injectedActivityConversi
onFactor)

res=linfit(x, y, YFIT=yfit)
slope=res(1)
intercept=res(0)

lineReg =  x*slope + intercept

plot, fullx, fully, /YSTYLE, psym=4, color = 0, background = 16777215, xtitle = '!MI!NC(s)dt /
C(t)', ytitle= 'C!IT!N(t)/C(t)', charsize = 2
oplot, x, lineReg, color = 0

endif
END

PRO LVA SliceData::LVA PlotSelectedVoxel
if(ptr valid(self.timeStampArrayPtr) eq 1 AND ptr valid(self.k1Ptr) eq 1 $
AND (self.selectedX gt self.windowXOffset OR self.selectedX eq self.windowXOffset) $
AND (self.selectedX gt self.windowXOffset OR self.selectedX eq self.windowXOffset) ) then
begin

; To avoid all the indexing we create some temporal variables
timestamp = *self.adjustedTimeStampArrayPtr                    ;Original time values

; Find the size of the time axis and create an extended one
nrTimestamps = (size(*(self.timeStampArrayPtr), /dimension))(1)
;time_interpol=dindgen(self.interpolFactor*( nrTimestamps))/self.interpolFactor
time_interpol       = dindgen(self.interpolFactor* max(timestamp))/self.interpolFactor

intensity = fltarr(nrTimestamps)
intensity = (*self.sliceArrayPtr)[self.selectedX, self.selectedY, self.depth, * ];Original
pixel values

k1 = float((*self.k1Ptr)[self.selectedX, self.selectedY, self.depth])
k2 = float((*self.k2Ptr)[self.selectedX, self.selectedY, self.depth])
k3 = float((*self.k3Ptr)[self.selectedX, self.selectedY, self.depth])
k4 = float((*self.k4Ptr)[self.selectedX, self.selectedY, self.depth])
vb = float((*self.vbPtr)[self.selectedX, self.selectedY, self.depth])
ps = float((*self.patSlopePtr)[self.selectedX, self.selectedY, self.depth])
pi = float((*self.patInterPtr)[self.selectedX, self.selectedY, self.depth])

r2 = 0
correl = 0

if( k1 eq 0 AND k2 eq 0 AND k3 eq 0 AND k4 eq 0) then begin
return
originalVoxelAdaption = fltarr(1)
voxelAdaption = fltarr(1)
boundVoxelAdaption = fltarr(1)
```

```
freeVoxelAdaption = fltarr(1)
vascularAdaption = fltarr(1)
endif else begin

intensity interpol  = double(interpol(*self.arteriePtr, *self.adjustedTimeStampArrayPtr,
time_interpol))

voxelAdaption = trekomp(time_interpol, [k1, k2, k3, k4, vb], plasmaFunc =
*self.arterieInterpolPtr )    ;Interpolated curve

boundVoxelAdaption = bound(time_interpol, [k1, k2, k3, k4, vb], *self.arterieInterpolPtr )
       ;Bound curve
freeVoxelAdaption = free(time_interpol, [k1, k2, k3, k4, vb], *self.arterieInterpolPtr ) ;Free
curve
vascularAdaption = (*self.arterieInterpolPtr)*vb
correl = (correlate(voxelAdaption, intensity))^2
*(self.currentVoxelAdaptationPtr) = voxelAdaption
endelse

SUV = (*self.sliceArrayPtr)[self.selectedX, self.selectedY, self.depth, self.time ] *
self.imageIntensityScale*self.weight*1000/(self.injectedActivity/self.injectedActivityConversi
onFactor)
SUVVoxel = intensity*
self.imageIntensityScale*self.weight*1000/(self.injectedActivity/self.injectedActivityConversi
onFactor)
SUVVoxelAdaption = voxelAdaption *
self.imageIntensityScale*self.weight*1000/(self.injectedActivity/self.injectedActivityConversi
onFactor)
SUVBound = boundVoxelAdaption *
self.imageIntensityScale*self.weight*1000/(self.injectedActivity/self.injectedActivityConversi
onFactor)
SUVFree = freeVoxelAdaption *
self.imageIntensityScale*self.weight*1000/(self.injectedActivity/self.injectedActivityConversi
onFactor)
SUVVascular = vascularAdaption *
self.imageIntensityScale*self.weight*1000/(self.injectedActivity/self.injectedActivityConversi
onFactor)

fc = self.foregroundColor
bc = self.backgroundColor
plot, timestamp, SUVVoxel, /YSTYLE, psym=4,color = fc , background = bc, xtitle = 'Time(min)',
ytitle = 'SUV', charsize = 2
oplot, time interpol, SUVVoxelAdaption, psym=-3,color = fc
oplot, time interpol, SUVBound, linestyle=2,color = fc
oplot, time_interpol, SUVFree, linestyle=3,color = fc
oplot, time_interpol, SUVVascular, linestyle=4,color = fc

legend, ['Bound','Free','Vascular'],linestyle=[2, 3, 4], color = [fc, fc, fc], textcolors =
[fc, fc, fc], charsize = 1.5, position = self.legendPosition, pspacing = 1
xyouts, self.textPosition[0], self.textPosition[1]+140,   'PS:  ' + strtrim(string(ps,
format='(F6.3)'), 1), /DEVICE, color = fc
xyouts, self.textPosition[0], self.textPosition[1]+120,   'PI:  ' + strtrim(string(pi,
format='(F6.3)'), 1), /DEVICE, color = fc
xyouts, self.textPosition[0], self.textPosition[1]+100,   'VB:  ' + strtrim(string(vb,
format='(F6.3)'), 1), /DEVICE, color = fc
xyouts, self.textPosition[0], self.textPosition[1]+80,    'R2:  ' + strtrim(string(correl,
format='(F6.2)'), 1), /DEVICE, color = fc
xyouts, self.textPosition[0], self.textPosition[1]+60,    'SUV: ' + strtrim(string(SUV,
format='(F6.2)'), 1), /DEVICE, color = fc
xyouts, self.textPosition[0], self.textPosition[1]+40,    'K1:  ' + strtrim(string(k1,
format='(F6.2)'), 1), /DEVICE, color = fc
xyouts, self.textPosition[0], self.textPosition[1]+20,    'K2:  ' + strtrim(string(k2,
format='(F6.2)'), 1), /DEVICE, color = fc
xyouts, self.textPosition[0], self.textPosition[1],       'K3:  ' + strtrim(string(k3,
format='(F6.3)'), 1), /DEVICE, color = fc
;stop
endif
END

PRO LVA SliceData::LVA PlotSelectedPatlak
if(ptr valid(self.timeStampArrayPtr) eq 1 $
AND (self.selectedX gt self.windowXOffset OR self.selectedX eq self.windowXOffset) $
AND (self.selectedX gt self.windowXOffset OR self.selectedX eq self.windowXOffset) ) then
begin

timeStampDimension = size(*(self.timeStampArrayPtr), /dimension)
```

```
nrTimestamps = timeStampDimension[1]
timestamp = *self.adjustedTimeStampArrayPtr

plasIntensity = fltarr(nrTimestamps)
for sequence = 0L, self.nrPlasmaPoints -1 do begin
for iter = 0L, nrTimestamps-1 DO BEGIN
plasIntensity[iter] = plasIntensity[iter] + (*self.sliceArrayPtr)[self.plasma[0, sequence],
self.plasma[1, sequence], self.plasma[2, sequence], iter ]
endfor
endfor
plasIntensity = plasIntensity / self.nrPlasmaPoints

art_kum = fltarr(nrTimestamps)
for i=1, nrTimestamps-1 do art_kum[i]= int_tabulated(timestamp(0:i), plasIntensity(0:i),
/DOUBLE)

fullx = art_kum/plasIntensity
fully = (*self.sliceArrayPtr)[self.selectedX, self.selectedY, self.depth, * ]/plasIntensity

time ind=where(timestamp ge 20)
x = art kum(time ind)/plasIntensity(time ind)
y = (*self.sliceArrayPtr)[self.selectedX, self.selectedY, self.depth, time_ind
]/plasIntensity(time_ind)
res=linfit(x, y, YFIT=yfit)
slope=res(1)
intercept=res(0)

lineReg =  x*slope + intercept

plot, fullx, fully, /YSTYLE, psym=4, color = 0, background = 16777215, xtitle = '!MI!NC(s)dt /
C(t)', ytitle= 'C!IT!N(t)/C(t)', charsize = 2
oplot, x, lineReg, color = 0
endif
END
```

Save load functions

```
PRO LVA_SliceData::LVA_SaveState, savefile
if(savefile eq '') then return
OPENW, exitFile, savefile, /GET_LUN
;ID tag
WRITEU, exitFile, float(1337.847)
;Source directory
WRITEU, exitFile, strlen(self.imageDirectoryPath)
WRITEU, exitFile, self.imageDirectoryPath

WRITEU, exitFile, strlen(self.ctImageDirectoryPath)
WRITEU, exitFile, self.ctImageDirectoryPath

;Zoom position and alignment shift
WRITEU, exitFile, float(self.windowXOffset)
WRITEU, exitFile, float(self.windowYOffset)
WRITEU, exitFile, float(self.manualCTShift[0])
WRITEU, exitFile, float(self.manualCTShift[1])
WRITEU, exitFile, float(self.manualCTShift[2])

;Calculation startPoints
WRITEU, exitFile, float(self.k1_init)
WRITEU, exitFile, float(self.k2_init)
WRITEU, exitFile, float(self.k3_init)
WRITEU, exitFile, float(self.k4_init)
WRITEU, exitFile, float(self.k1 limited[0])
WRITEU, exitFile, float(self.k1 limits[0])
WRITEU, exitFile, float(self.k1_limited[1])
WRITEU, exitFile, float(self.k1_limits[1])
WRITEU, exitFile, float(self.k2_limited[0])
WRITEU, exitFile, float(self.k2 limits[0])
WRITEU, exitFile, float(self.k2 limited[1])
WRITEU, exitFile, float(self.k2_limits[1])
WRITEU, exitFile, float(self.k3_limited[0])
WRITEU, exitFile, float(self.k3_limits[0])
```

```
WRITEU, exitFile, float(self.k3_limited[1])
WRITEU, exitFile, float(self.k3_limits[1])
WRITEU, exitFile, float(self.k4 limited[0])
WRITEU, exitFile, float(self.k4 limits[0])
WRITEU, exitFile, float(self.k4 limited[1])
WRITEU, exitFile, float(self.k4 limits[1])
WRITEU, exitFile, float(self.relstep)
WRITEU, exitFile, float(self.minPlasmaRatio)
WRITEU, exitFile, float(self.minPlasmaRatioLimited)

;Nr of plasma elements
WRITEU, exitFile, float(self.nrPlasmaPoints)
;Element data
for index = 0, self.nrPlasmaPoints-1 do begin
WRITEU, exitFile, self.Plasma[0, index]
WRITEU, exitFile, self.Plasma[1, index]
WRITEU, exitFile, self.Plasma[2, index]
WRITEU, exitFile, self.Plasma[3, index]
endfor

;Nr of tumor elements
WRITEU, exitFile, float(self.nrTumorPoints)
;Element data
for index = 0, self.nrTumorPoints-1 do begin
WRITEU, exitFile, self.Tumor[0, index]
WRITEU, exitFile, self.Tumor[1, index]
WRITEU, exitFile, self.Tumor[2, index]
WRITEU, exitFile, self.Tumor[3, index]
endfor

;Nr of tissue elements
WRITEU, exitFile, float(self.nrTissuePoints)
;Element data
for index = 0, self.nrTissuePoints-1 do begin
WRITEU, exitFile, self.Tissue[0, index]
WRITEU, exitFile, self.Tissue[1, index]
WRITEU, exitFile, self.Tissue[2, index]
WRITEU, exitFile, self.Tissue[3, index]
endfor

WRITEU, exitFile, float(self.windowXpos)
WRITEU, exitFile, float(self.windowYpos)

;Output windows
WRITEU, exitFile, float(self.windowTumorXpos)
WRITEU, exitFile, float(self.windowTumorYpos)
WRITEU, exitFile, float(self.windowTumorXsize)
WRITEU, exitFile, float(self.windowTumorYsize)
WRITEU, exitFile, float(self.windowOutputXpos)
WRITEU, exitFile, float(self.windowOutputYpos)
WRITEU, exitFile, float(self.windowOutputXsize)
WRITEU, exitFile, float(self.windowOutputYsize)

;Class type
WRITEU, exitFile, float(self.tumorLocationCode)
WRITEU, exitFile, float(self.tumorTypeCode)
WRITEU, exitFile, float(self.yearOfBirth)

;Output ranges
WRITEU, exitFile, float(self.petDisplayRange[0])
WRITEU, exitFile, float(self.petDisplayRange[1])
WRITEU, exitFile, float(self.K1DisplayRange[0])
WRITEU, exitFile, float(self.K1DisplayRange[1])
WRITEU, exitFile, float(self.k2DisplayRange[0])
WRITEU, exitFile, float(self.k2DisplayRange[1])
WRITEU, exitFile, float(self.k3DisplayRange[0])
WRITEU, exitFile, float(self.k3DisplayRange[1])
WRITEU, exitFile, float(self.mbDisplayRange[0])
WRITEU, exitFile, float(self.mbDisplayRange[1])
WRITEU, exitFile, float(self.vbDisplayRange[0])
WRITEU, exitFile, float(self.vbDisplayRange[1])
;self->PrintThresholds
CLOSE, exitFile
FREE_LUN, exitFile
END
```

```
PRO LVA_SliceData::LVA_LoadState, savefile
;Load exit state
if(savefile eq '' OR self.locked eq 1) then return
progressBar = Obj New("PROGRESSBAR")
progressBar -> Start
progressBar -> SetProperty, Text='Loading lva file'

OPENR, exitFile, savefile, /GET_LUN, ERROR = err

if(err eq 0) then begin
if(EOF(exitFile) ne 1) then begin
ReadU, exitFile, version
if(version gt 1337.844) then begin
dirLength = 1L
ReadU, exitfile, dirLength
if(dirLength eq 0) then return

imdir = String(Replicate(32B, long(dirLength)))
ReadU, exitFile, imDir
;radiumBase = 'C:\\Temp\\Pas\(Espen\)'
;homeBase = 'C:\kingback\PAS'
;radiumBase = 'C:\Temp\Pas(Espen)'
radiumBase = 'C:\Temp\pas'
homeBase = 'C:\\kingback\\PAS'
;imDir = strjoin(strsplit(imDir, homeBase, /regex, /extract, /preserve_null), radiumBase)
;                    stop
self->LVA LoadDirectory, imDir
progressBar -> Update, 50
dirLength = 1L
ReadU, exitfile, dirLength
if(dirLength ne 0) then begin
imdir = String(Replicate(32B, long(dirLength)))
ReadU, exitFile, imDir
;imDir = strjoin(strsplit(imDir, homeBase, /regex, /extract, /preserve_null), radiumBase)
self->LVA_LoadCTDirectory, imDir
endif
progressBar -> Update, 90
;Zoom position and alignment shift
ReadU, exitFile, wx
ReadU, exitFile, wy
self.windowXOffset = wx
self.windowYOffset = wy
ReadU, exitFile, manx
ReadU, exitFile, many
ReadU, exitFile, manz
self->LVA_SetManualCTShift, manx, many

;Calculation startPoints
ReadU, exitFile, k1in
ReadU, exitFile, k2in
ReadU, exitFile, k3in
ReadU, exitFile, k4in
self->LVA_SetInitialKEstimates, k1in, k2in, k3in, k4in
ReadU, exitFile, k1Liml
ReadU, exitFile, k1Vall
ReadU, exitFile, k1Limh
ReadU, exitFile, k1Valh
ReadU, exitFile, k2Liml
ReadU, exitFile, k2Vall
ReadU, exitFile, k2Limh
ReadU, exitFile, k2Valh
ReadU, exitFile, k3Liml
ReadU, exitFile, k3Vall
ReadU, exitFile, k3Limh
ReadU, exitFile, k3Valh
ReadU, exitFile, k4Liml
ReadU, exitFile, k4Vall
ReadU, exitFile, k4Limh
ReadU, exitFile, k4Valh

if(k2Limh ne 1. OR k2Valh ne 3.5 OR k3Limh ne 1. OR k3Valh ne 0.75) then begin
print, 'Unacceptable preference value ', k2Limh, k2Valh, k3Limh, k3Valh
k2Limh = 1.
k2Valh = 3.5
k3Limh = 1.
k3Valh = 0.75
```

```
endif
self->LVA_SetKLimits, [k1Liml, k1Limh], [k1Val1, k1Valh], [k2Liml, k2Limh], [k2Val1, k2Valh],
[k3Liml, k3Limh], [k3Val1, k3Valh], [k4Liml, k4Limh], [k4Val1, k4Valh]

ReadU, exitFile, relstep
ReadU, exitFile, minpratio
ReadU, exitFile, mpratiolim
self->LVA_SetRelstep, relstep
self->LVA SetPlasmaRatioLimited, mpratiolim
self->LVA_SetMinPlasmaRatio, minpratio

;The Roi's
nrPoints = 0.0
xpos = 0.0
ypos = 0.0
depth = 0.0
time = 0.0

;Save original values
originalDepth  = self.depth
originalTime = self.time

;Load plasma points
self.nrPlasmaPoints = 0L
self.nrTumorPoints = 0L
self.nrTissuePoints = 0L

if(EOF(exitFile) ne 1) then begin ReadU, exitFile, nrPoints
endif else begin print, 'File empty'
endelse
for plindex = 0, nrPoints -1 do begin
if(EOF(exitFile) ne 1) then ReadU, exitFile, xpos
if(EOF(exitFile) ne 1) then ReadU, exitFile, ypos
if(EOF(exitFile) ne 1) then ReadU, exitFile, depth
if(EOF(exitFile) ne 1) then ReadU, exitFile, time
self.depth = depth
self.time = time
if(self.nrPlasmaPoints lt self.maxNrPlasmaPoints-1) then begin
self.plasma[0, self.nrPlasmaPoints] = xpos
self.plasma[1, self.nrPlasmaPoints] = ypos
self.plasma[2, self.nrPlasmaPoints] = depth
self.plasma[3, self.nrPlasmaPoints] = time
self.nrPlasmaPoints = self.nrPlasmaPoints +1
endif
endfor

;Load tumor points
if(EOF(exitFile) ne 1) then begin ReadU, exitFile, nrPoints
endif else begin print, 'File empty'
endelse
for plindex = 0, nrPoints -1 do begin
if(EOF(exitFile) ne 1) then ReadU, exitFile, xpos
if(EOF(exitFile) ne 1) then ReadU, exitFile, ypos
if(EOF(exitFile) ne 1) then ReadU, exitFile, depth
if(EOF(exitFile) ne 1) then ReadU, exitFile, time
self.depth = depth
self.time = time
if(self.nrTumorPoints lt self.maxNrTumorPoints-1) then begin
self.tumor[0, self.nrTumorPoints] = xpos
self.tumor[1, self.nrTumorPoints] = ypos
self.tumor[2, self.nrTumorPoints] = depth
self.tumor[3, self.nrTumorPoints] = time
self.nrTumorPoints = self.nrTumorPoints +1
endif
endfor


;Load tissue points
if(EOF(exitFile) ne 1) then begin ReadU, exitFile, nrPoints
endif else begin print, 'File empty'
endelse

for plindex = 0, nrPoints -1 do begin
if(EOF(exitFile) ne 1) then ReadU, exitFile, xpos
if(EOF(exitFile) ne 1) then ReadU, exitFile, ypos
if(EOF(exitFile) ne 1) then ReadU, exitFile, depth
```

```
if(EOF(exitFile) ne 1) then ReadU, exitFile, time
self.depth = depth
self.time = time
if(self.nrTissuePoints lt self.maxNrTissuePoints-1) then begin
self.tissue[0, self.nrTissuePoints] = xpos
self.tissue[1, self.nrTissuePoints] = ypos
self.tissue[2, self.nrTissuePoints] = depth
self.tissue[3, self.nrTissuePoints] = time
self.nrTissuePoints = self.nrTissuePoints +1
endif
endfor


;Load viewport center
if(EOF(exitFile) ne 1) then begin ReadU, exitFile, xpos
endif else begin print, 'File empty'
endelse
if(EOF(exitFile) ne 1) then begin ReadU, exitFile, ypos
endif else begin print, 'File empty'
endelse

;print, 'Read ', xpos, ' ', ypos
self.windowXpos = xpos
self.windowYpos = ypos
self.windowXOffset = xpos
self.windowYOffset = ypos

if(version gt 1337.845) then begin
;Output windows
if(EOF(exitFile) ne 1) then ReadU, exitFile, tumorXpos
if(EOF(exitFile) ne 1) then ReadU, exitFile, tumorYpos
if(EOF(exitFile) ne 1) then ReadU, exitFile, tumorXsize
if(EOF(exitFile) ne 1) then ReadU, exitFile, tumorYsize
if(EOF(exitFile) ne 1) then ReadU, exitFile, outputXpos
if(EOF(exitFile) ne 1) then ReadU, exitFile, outputYpos
if(EOF(exitFile) ne 1) then ReadU, exitFile, outputXsize
if(EOF(exitFile) ne 1) then ReadU, exitFile, outputYsize

self.windowTumorXpos = tumorXpos
self.windowTumorYpos = tumorYpos
self.windowTumorXsize = tumorXsize
self.windowTumorYsize = tumorYsize
self.windowOutputXpos = outputXpos
self.windowOutputYpos = outputYpos
self.windowOutputXsize = outputXsize
self.windowOutputYsize = outputYsize

;Class type
if(EOF(exitFile) ne 1) then ReadU, exitFile, tumLocCode
if(EOF(exitFile) ne 1) then ReadU, exitFile, tumTypCode
if(EOF(exitFile) ne 1) then begin ReadU, exitFile, ybirth
endif else print, 'File empty'
self.tumorLocationCode = tumLocCode
self.tumorTypeCode = tumTypCode
self.yearOfBirth = ybirth
endif

if(version gt 1337.846) then begin
if(EOF(exitFile) ne 1) then ReadU, exitFile, petMin
self.petDisplayRange[0] = float(petMin)
if(EOF(exitFile) ne 1) then ReadU, exitFile, petMax
self.petDisplayRange[1] = float(petMax)
if(EOF(exitFile) ne 1) then ReadU, exitFile, k1Min
self.K1DisplayRange[0] = float(k1Min)
if(EOF(exitFile) ne 1) then ReadU, exitFile, k1Max
self.K1DisplayRange[1] = float(k1Max)
if(EOF(exitFile) ne 1) then ReadU, exitFile, k2Min
self.k2DisplayRange[0] = float(k2Min)
if(EOF(exitFile) ne 1) then ReadU, exitFile, k2Max
self.k2DisplayRange[1] = float(k2Max)
if(EOF(exitFile) ne 1) then ReadU, exitFile, k3Min
self.k3DisplayRange[0] = float(k3Min)
if(EOF(exitFile) ne 1) then ReadU, exitFile, k3Max
self.k3DisplayRange[1] = float(k3Max)
if(EOF(exitFile) ne 1) then ReadU, exitFile, mbMin
self.mbDisplayRange[0] = float(mbMin)
```

```
if(EOF(exitFile) ne 1) then ReadU, exitFile, mbMax
self.mbDisplayRange[1] = float(mbMax)
if(EOF(exitFile) ne 1) then ReadU, exitFile, vbMin
self.vbDisplayRange[0] = float(vbMin)
if(EOF(exitFile) ne 1) then ReadU, exitFile, vbMax
self.vbDisplayRange[1] = float(vbMax)
;self->PrintThresholds
endif
;restore original values
self.depth = originalDepth
self.time = originalTime
endif else begin
print, 'Unsuported savefile'
endelse
endif
endif
if progressBar -> CheckCancel() then self.locked = 1

CLOSE, exitFile
FREE_LUN, exitFile
progressBar -> Destroy
print, 'Loaded ' + savefile
END

PRO LVA_SliceData::PrintThresholds
print, self.petDisplayRange
print, self.K1DisplayRange
print, self.k2DisplayRange
print, self.k3DisplayRange
print, self.mbDisplayRange
print, self.vbDisplayRange
END

PRO LVA_SliceData::LVA_WriteKHistogram, KvaluePtr, binsize, saveFile
WRITEU, saveFile, float(binsize)
if(max(*KvaluePtr) eq 0) then begin
WRITEU, saveFile, 0
endif else begin
kindex = where(*KvaluePtr gt 0)
kvalues = floor(((*KvaluePtr)[kindex])/binSize)
;sortedK = kvalues(sort(kvalues))
nrBins = max(kvalues)+1 ;size(uniq(sortedK), /dimensions)
kHistogram = intarr(nrBins)
nrValues = (size(kvalues, /dimensions))[0]
for ind = 0, (nrValues-2) do begin
binValue = kvalues[ind]
kHistogram[binValue]++
endfor

WRITEU, saveFile, float(nrBins)
for index = 0, nrBins-1 do begin
WRITEU, saveFile, float(kHistogram[index])
endfor
endelse
END

PRO LVA_SliceData::LVA_WriteValuegram, valuePtr, saveFile

if(max(*valuePtr) eq 0) then begin
WRITEU, saveFile, 0.0
endif else begin
WRITEU, saveFile, float(self.nrTumorPoints)
for tpoint = 0L, self.nrTumorPoints -1 do begin
WRITEU, saveFile, float((*valuePtr)[self.tumor[0, tpoint], self.tumor[1, tpoint],
self.tumor[2, tpoint] ])
endfor
endelse
END

PRO LVA_SliceData::LVA_SaveTrace, savefile
if(savefile eq '') then return
OPENW, traceFile, savefile, /GET_LUN
;ID tag
WRITEU, traceFile, float(10054.036)

self->LVA_WriteValuegram, self.k1ptr, traceFile
```

```
self->LVA_WriteValuegram, self.k2ptr, traceFile
self->LVA_WriteValuegram, self.k3ptr, traceFile
self->LVA WriteValuegram, self.VBPtr, traceFile
self->LVA WriteValuegram, self.pearsonPtr, traceFile
self->LVA WriteValuegram, self.patSlopePtr, traceFile
self->LVA_WriteValuegram, self.patInterPtr, traceFile

WRITEU, traceFile, float(self.imageIntensityScale)
WRITEU, traceFile, float(self.weight)
WRITEU, traceFile, float(self.injectedActivity/self.injectedActivityConversionFactor)
WRITEU, traceFile, float(self.tumorLocationCode)
WRITEU, traceFile, float(self.tumorTypeCode)
WRITEU, traceFile, float(self.yearOfBirth)
WRITEU, traceFile, float(abs((*self.depthStampArrayPtr)[0] - (*self.depthStampArrayPtr)[1]))

;Write plasma array
nrplasmaElements = (size((*self.arteriePtr), /dimensions))[0]
;Nr of plasma elements
WRITEU, traceFile, float(nrplasmaElements)
;Element data
for index = 0, nrplasmaElements-1 do begin
WRITEU, traceFile, float((*self.arteriePtr)[index])
endfor

;Write time array
timeStampDimension = size(*(self.timeStampArrayPtr), /dimension)
nrTimestamps = timeStampDimension[1]
timestamp = *self.adjustedTimeStampArrayPtr
WRITEU, traceFile, float(nrTimestamps)
for index = 0, nrTimestamps-1 do begin
WRITEU, traceFile, float(timestamp[index])
endfor

;Write tumor histogram
tumorBinSize = 0.1
tumorValues = fltarr(self.nrTumorPoints ,nrTimestamps)

SUVMap = (*self.sliceArrayPtr)*
self.imageIntensityScale*self.weight*1000.0/(self.injectedActivity/self.injectedActivityConver
sionFactor)

WRITEU, traceFile, float(self.nrTumorPoints)
WRITEU, traceFile, float(nrTimestamps)
for tpoint = 0L, self.nrTumorPoints -1 do begin
for iter = 0L, nrTimestamps-1 DO BEGIN
WRITEU, traceFile, float(SUVMap[self.tumor[0, tpoint], self.tumor[1, tpoint], self.tumor[2,
tpoint], iter ])
endfor
endfor

; Write peak value
for tpoint = 0L, self.nrTumorPoints -1 do begin
for iter = 0L, nrTimestamps-1 DO BEGIN
accum = SUVMap[self.tumor[0, tpoint], self.tumor[1, tpoint], self.tumor[2, tpoint], iter ]/7.
accum += SUVMap[self.tumor[0, tpoint]+1, self.tumor[1, tpoint], self.tumor[2, tpoint], iter
]/7.
accum += SUVMap[self.tumor[0, tpoint]-1, self.tumor[1, tpoint], self.tumor[2, tpoint], iter
]/7.
accum += SUVMap[self.tumor[0, tpoint], self.tumor[1, tpoint]+1, self.tumor[2, tpoint], iter
]/7.
accum += SUVMap[self.tumor[0, tpoint], self.tumor[1, tpoint]-1, self.tumor[2, tpoint], iter
]/7.
accum += SUVMap[self.tumor[0, tpoint], self.tumor[1, tpoint], self.tumor[2, tpoint]+1, iter
]/7.
accum += SUVMap[self.tumor[0, tpoint], self.tumor[1, tpoint], self.tumor[2, tpoint]-1, iter
]/7.
WRITEU, traceFile, float(accum)
endfor
endfor

;Write Tissue values
WRITEU, traceFile, float(self.nrTissuePoints)
for tpoint = 0L, self.nrTissuePoints -1 do begin
for iter = 0L, nrTimestamps-1 DO BEGIN
WRITEU, traceFile, float(SUVMap[self.tissue[0, tpoint], self.tissue[1, tpoint], self.tissue[2,
tpoint], iter ])
```

```
endfor
endfor

;Write Patlak values
patlak = self->LVA CalculatePatlak()
patDim = size(patlak, /dimensions)
WRITEU, traceFile, float(patDim[0])
WRITEU, traceFile, float(patDim[1])

for arraynr = 0L, patDim[0] -1 do begin
for valuenr = 0L, patDim[1]-1 DO BEGIN
WRITEU, traceFile, float(patlak[arraynr, valuenr])
endfor
endfor
negative = 0
WRITEU, traceFile, float(self.nrTumorPoints)
for tumorPoint = 0L, self.nrTumorPoints -1 do begin
k1 = (*self.k1Ptr)[self.tumor[0, tumorPoint], self.tumor[1, tumorPoint], self.tumor[2,
tumorPoint]]
k2 = (*self.k2Ptr)[self.tumor[0, tumorPoint], self.tumor[1, tumorPoint], self.tumor[2,
tumorPoint]]
k3 = (*self.k3Ptr)[self.tumor[0, tumorPoint], self.tumor[1, tumorPoint], self.tumor[2,
tumorPoint]]
if(float(k1*k3/float(k2+k3)) lt 0) then begin
negative++
if(negative lt 2) then stop
endif
WRITEU, traceFile,  float(k1*k3/float(k2+k3))
endfor
if(negative gt 0) then print, 'Found ', negative, ' negative values of metabolic rate'

WRITEU, traceFile, float(3703)

CLOSE, traceFile
FREE_LUN, traceFile
END

PRO LVA_SliceData::LVA_LoadDirectory, dir
self.imageDirectoryPath = dir
if(self.imageDirectoryPath eq '') then return
files = FILE_SEARCH(self.imageDirectoryPath, '*', COUNT=nimages, /NOSORT)
if(nimages eq 0) then return

myDicom = OBJ_NEW('IDLffDICOM')
read = myDicom->Read(files[0])
dimXYptr = myDicom->GetValue('0028'x, '0010'x, /NO_COPY)
dim_xy = *dimXYptr[0]
resZptr= myDicom->GetValue('0018'x, '0050'x, /NO_COPY)
res_z = float(strcompress(strmid(*resZptr[0], 0,6), /REMOVE_ALL))
resXYptr= myDicom->GetValue('0028'x, '0030'x, /NO_COPY)
res_xy = float(strcompress(strmid(*resXYptr[0], 0,6), /REMOVE_ALL))
self.petSliceResolution(0) = res_xy
self.petSliceResolution(1) = res_xy
self.petSliceResolution(2) = res_z
ScaleValuePtr = myDicom->GetValue('0028'x, '1053'x, /NO_COPY)
self.imageIntensityScale = float(*ScaleValuePtr[0])
posPtr = myDicom->GetValue('0020'x, '0032'x, /NO_COPY)
tmpPos=strsplit(*posPtr[0], '\', /EXTRACT)
self.petSlicePosition(0) = float(tmpPos[0])
self.petSlicePosition(1) = float(tmpPos[1])
weightptr = myDicom->GetValue('0010'x, '1030'x, /NO_COPY)
self.weight = *weightptr[0]
activityptr = myDicom->GetValue('0018'x, '1074'x, /NO_COPY)
self.injectedActivity = *activityptr[0]
if(self.injectedActivity gt 1000000000) then self.injectedActivityConversionFactor = 37.0$
else self.injectedActivityConversionFactor = 1.0
self.SUVConversionFactor =
self.imageIntensityScale*self.weight*1000.0/(self.injectedActivity/self.injectedActivityConver
sionFactor)
myDicom->Reset

;Temporal arrays
zpos=fltarr(nimages)
series_instance=intarr(nimages)
acquisition_time=fltarr(nimages)
```

```
;Analyze structure
for i=0L, nimages-1L do begin
read = myDicom->Read(files[i])
posPtr = myDicom->GetValue('0020'x, '0032'x, /NO_COPY)
if(ptr_valid(posPtr) ne 1) then begin
print, 'Invalid dcm file detected'
break
endif
tmpPos=strsplit(*posPtr[0], '\', /EXTRACT)
zpos[i]=float(tmpPos[2])
instancePtr = myDicom->GetValue('0020'x, '0013'x, /NO_COPY)
series_instance[i]=*instancePtr[0]

acquisitionPtr = myDicom->GetValue('0008'x, '0032'x, /NO_COPY)
acquisition time[i]=float(strmid(*acquisitionPtr[0], 0, 2))+(float(strmid(*acquisitionPtr[0],
2, 2))/60.0)+(float(strmid(*acquisitionPtr[0], 4, 2))/3600)

myDicom->Reset
endfor

sort_ind=MULTISORT(zpos, acquisition_time, series_instance)

self.petSlicePosition(2) = float(zpos[sort_ind[0]])

dim_z = size(uniq(zpos[sort_ind]), /dimensions)
dim_t = long(nimages/dim_z)

;Allocate buffers
self->LVA_EraseBuffers
self.sliceArrayPtr = ptr_new(fltarr(dim_xy, dim_xy, dim_z, dim_t))
self.timeStampArrayPtr = ptr_new(fltarr(dim_z, dim_t))
self.adjustedTimeStampArrayPtr = ptr_new(fltarr(dim t))
self.depthStampArrayPtr = ptr new(fltarr(dim z, dim t))
self.k1Ptr      = ptr new(dblarr(dim xy, dim xy, dim z))
self.k2Ptr      = ptr_new(dblarr(dim_xy, dim_xy, dim_z))
self.k3Ptr      = ptr_new(dblarr(dim_xy, dim_xy, dim_z))
self.k4Ptr      = ptr_new(dblarr(dim_xy, dim_xy, dim_z))
self.VBPtr      = ptr new(dblarr(dim xy, dim xy, dim z))
self.pearsonPtr = ptr new(dblarr(dim xy, dim xy, dim z))
self.patSlopePtr   = ptr new(dblarr(dim xy, dim xy, dim z))
self.patInterPtr   = ptr_new(dblarr(dim_xy, dim_xy, dim_z))
self.arteriePtr = ptr_new(fltarr(dim_t))
self.arterieInterpolPtr = ptr new(fltarr(dim t*self.interpolFactor))
self.currentVoxelAdaptationPtr = ptr new(fltarr(dim t*self.interpolFactor))
self.timeInterpolPtr = ptr_new(dblarr(dim_t*self.interpolFactor))

;scaleValue = 0.0d
; Write the new data to the buffers

for i=0L, nimages-1 do begin
index=sort_ind(i)
timeIndex = i MOD dim_t
zIndex = i / dim_t
read = myDicom->Read(files[index])
imageDataPtr = myDicom->GetValue('7fe0'x, '0010'x, /NO COPY)
;newScaleValuePtr = myDicom->GetValue('0028'x, '1053'x, /NO COPY)
;scaleValue = *newScaleValuePtr[0]  ;There is only one scale, store as array otherwise
;print, size(imageDataPtr)
;print, highResImage
highResImage = (size(imageDataPtr, /dimensions))[0] -1 ;Identify if there exists a high
resolution set
;if(highResImage ne 1) then print, 'Found only ' + string(highResImage+1) + ' resolution'
(*self.sliceArrayPtr)[*,*, zIndex, timeIndex] =
rebin(rotate(float(*imageDataPtr[highResImage]), 7), dim_xy, dim_xy)
(*self.timeStampArrayPtr)[zIndex, timeIndex] = acquisition time[index]
(*self.depthStampArrayPtr)[zIndex, timeIndex] = zpos[index]

myDicom->Reset
endfor
self.bufferEmpty = 0L
OBJ_DESTROY, myDicom

self.depth = 0L
self.time = 0L

;Create the time vector starting at 0
```

```
if(size(*self.timeStampArrayPtr, /n_dimensions) gt 1) then begin
*(self.adjustedTimeStampArrayPtr) = ((*self.timeStampArrayPtr)[0,*] -
(*self.timeStampArrayPtr)[0,0])*60
endif else *(self.adjustedTimeStampArrayPtr) = ((*self.timeStampArrayPtr) -
(*self.timeStampArrayPtr)[0])*60

*(self.timeInterpolPtr) = dindgen(self.interpolFactor*
max(*self.adjustedTimeStampArrayPtr))/self.interpolFactor
END

PRO LVA_SliceData::LVA_LoadCTDirectory, dir
self.ctImageDirectoryPath = dir
if(self.ctImageDirectoryPath eq '') then return
files = FILE_SEARCH(self.ctImageDirectoryPath, '*', COUNT=nimages, /NOSORT)
if(nimages eq 0) then return

myDicom = OBJ_NEW('IDLffDICOM')
read = myDicom->Read(files[0])
dimXYptr = myDicom->GetValue('0028'x, '0010'x, /NO_COPY)
dim_xy = *dimXYptr[0]
resZptr= myDicom->GetValue('0018'x, '0050'x, /NO_COPY)
res_z = float(strcompress(strmid(*resZptr[0], 0,6), /REMOVE_ALL))
resXYptr= myDicom->GetValue('0028'x, '0030'x, /NO_COPY)
res_xy = float(strcompress(strmid(*resXYptr[0], 0,6), /REMOVE_ALL))
self.ctImageResolution(0) = res_xy
self.ctImageResolution(1) = res_xy
self.ctImageResolution(2) = res_z
posPtr = myDicom->GetValue('0020'x, '0032'x, /NO_COPY)
tmpPos=strsplit(*posPtr[0], '\', /EXTRACT)
self.ctImageCenterPoint(0)=float(tmpPos[0])
self.ctImageCenterPoint(1)=float(tmpPos[1])
myDicom->Reset

;Temporal arrays
zpos=fltarr(nimages)

;Analyze structure
for i=0L, nimages-1L do begin
read = myDicom->Read(files[i])
posPtr = myDicom->GetValue('0020'x, '0032'x, /NO_COPY)
tmpPos=strsplit(*posPtr[0], '\', /EXTRACT)
zpos[i]=float(tmpPos[2])

myDicom->Reset
endfor

sort_ind=MULTISORT(zpos)

self.ctImageCenterPoint[2] = zpos[sort_ind[0]]

dim_z = size(uniq(zpos[sort_ind]), /dimensions)

if(ptr_valid(self.ctSliceArrayPtr)) then ptr_free, self.ctSliceArrayPtr
self.ctSliceArrayPtr = ptr_new(fltarr(dim_xy, dim_xy, dim_z))

if(ptr_valid(self.ctDepthStampArrayPtr)) then ptr_free, self.ctDepthStampArrayPtr
self.ctDepthStampArrayPtr = ptr_new(fltarr(dim_z))

; Write the new data to the buffers
for i=0L, nimages-1 do begin
index=sort ind(i)
read = myDicom->Read(files[index])
imageDataPtr = myDicom->GetValue('7fe0'x, '0010'x, /NO_COPY)
highResImage = (size(imageDataPtr, /dimensions))[0] -1
(*self.ctSliceArrayPtr)[*,*, i] = rebin(rotate((*imageDataPtr[highResImage]), 7), dim_xy,
dim xy)
(*self.ctDepthStampArrayPtr)[i] = zpos[index]
myDicom->Reset
endfor
OBJ_DESTROY, myDicom

END
```

Compartment functions

```
Function arterie, x, par
a=par(0)
b=par(1)
c=par(2)
d=par(3)
F=a*exp(-b*x)+c*exp(-d*x)
return, F
end

Function trekomp, x, par, plasmaFunc = plasmaFunc
F=free(x, par, plasmaFunc)+bound(x,par, plasmaFunc)
F=F+plasmaFunc*par(4)
return, F
end

Function free, x, par, plasma, debug = debug
if(N_ELEMENTS(debug) eq 0) then debug = 0
k1=double(par(0))
k2=double(par(1))
k3=double(par(2))
k4=double(par(3))

alfa1=0.5*(k2+k3+k4-sqrt(((k2+k3+k4)^2)-4*k2*k4))
alfa2=0.5*(k2+k3+k4+sqrt(((k2+k3+k4)^2)-4*k2*k4))

nel=n_elements(x)
F=dblarr(nel)

F1_h=(k1/(alfa2-alfa1))*(k4-alfa1)*exp(double(-alfa1*x(1:*)))*total((x(1:*)-x(0:nel-
2))*plasma(1:*)*exp(double(alfa1*x(1:*))), /CUMULATIVE, /DOUBLE)
F2_h=(k1/(alfa2-alfa1))*(alfa2-k4)*exp(double(-alfa2*x(1:*)))*total((x(1:*)-x(0:nel-
2))*plasma(1:*)*exp(double(alfa2*x(1:*))), /CUMULATIVE, /DOUBLE)

F1_v=(k1/(alfa2-alfa1))*(k4-alfa1)*exp(double(-alfa1*x(0:nel-2)))*total((x(1:*)-x(0:nel-
2))*plasma(0:nel-2)*exp(double(alfa1*x(0:nel-2))), /CUMULATIVE, /DOUBLE)
F2_v=(k1/(alfa2-alfa1))*(alfa2-k4)*exp(double(-alfa2*x(0:nel-2)))*total((x(1:*)-x(0:nel-
2))*plasma(0:nel-2)*exp(double(alfa2*x(0:nel-2))), /CUMULATIVE, /DOUBLE)

F(1:*)=0.5*(F1_h+F1_v+F2_h+F2_v)
return, F
end

Function bound, x, par, plasma, debug = debug
if(N_ELEMENTS(debug) eq 0) then debug = 0
k1=double(par(0))
k2=double(par(1))
k3=double(par(2))
k4=double(par(3))

alfa1=0.5*(k2+k3+k4-sqrt(((k2+k3+k4)^2)-4*k2*k4))
alfa2=0.5*(k2+k3+k4+sqrt(((k2+k3+k4)^2)-4*k2*k4))

nel=n_elements(x)
F=dblarr(nel)

if(alfa2 eq alfa1) then print, 'Alfa 1 and 2 equal, divide by zero'
fak=k1*k3/(alfa2-alfa1)

F1_h=exp(double(-alfa1*x(1:*)))*total((x(1:*)-x(0:nel-
2))*plasma(1:*)*exp(double(alfa1*x(1:*))), /CUMULATIVE, /DOUBLE)
F2_h=exp(double(-alfa2*x(1:*)))*total((x(1:*)-x(0:nel-
2))*plasma(1:*)*exp(double(alfa2*x(1:*))), /CUMULATIVE, /DOUBLE)

F1_v=exp(double(-alfa1*x(0:nel-2)))*total((x(1:*)-x(0:nel-2))*plasma(0:nel-
2)*exp(double(alfa1*x(0:nel-2))), /CUMULATIVE, /DOUBLE)
F2_v=exp(double(-alfa2*x(0:nel-2)))*total((x(1:*)-x(0:nel-2))*plasma(0:nel-
2)*exp(double(alfa2*x(0:nel-2))), /CUMULATIVE, /DOUBLE)

F(1:*)=fak*0.5*(F1_h+F1_v-F2_h-F2_v)
if(debug eq 1) then stop

return, F
end

function lhex2dec,inp
    out = 0L
```

```
    n = strlen(inp)
    for i=n-1,0,-1 do begin
      c = strupcase(strmid(inp,i,1))
      case c of
       'A': c = 10
       'B': c = 11
       'C': c = 12
       'D': c = 13
       'E': c = 14
       'F': c = 15
       else:
      endcase
      out = out + long(c)*16L^long(n-1-i)
    endfor
    return, out
end

PRO lplot, xarr, yarr, lOverWrite = myoplot, lcolor = mycolor, clear, _EXTRA = e
nrvalues = n_elements(xarr)
if(nrvalues lt 1) then return
if(KEYWORD_SET(myoplot) eq 0) then myoplot = 0
if(KEYWORD_SET(mycolor)) then begin
if( nrvalues eq n_elements(yarr) AND nrvalues eq n_elements(mycolor)) then begin
if(myoplot eq 0) then begin
plot, [xarr[0]], [yarr[0]], COLOR = mycolor[0], _EXTRA  = e
for index = 1, nrvalues-1 do begin
oplot, [xarr[index]], [yarr[index]], COLOR = mycolor[index], _EXTRA  = e
endfor
endif else begin
for index = 0, nrvalues-1 do begin
oplot, [xarr[index]], [yarr[index]], COLOR = mycolor[index], _EXTRA  = e
endfor
endelse
endif else print, 'Inconsistent array dimensions in lplot'
endif else if(myoplot eq 0) then plot, [xarr], [yarr], _EXTRA  = e else oplot, [xarr], [yarr],
_EXTRA  = e
END
```

Analyzer GUI

```
PRO ltgui_event, EVENT
if(TAG_NAMES(event, /STRUCTURE_NAME) EQ 'WIDGET_CONTEXT') then begin
print, 'context trap'
return
endif
widget_control, event.top, get_uvalue=dataPtr
widget_control, event.id, get_uvalue=widget

if(ptr_valid(dataPtr) eq 0) then print, 'Corrupt data pointer'

case widget of

'primaryWindow' : begin
if(event.type eq 0 AND event.press eq 4) then begin
WIDGET_DISPLAYCONTEXTMENU, event.ID, event.X, event.y, (*dataPtr).primaryContextBase
endif
endcase

'secondaryWindow' : begin
if(event.type eq 0 AND event.press eq 4) then begin
WIDGET_DISPLAYCONTEXTMENU, event.ID, event.X, event.y, (*dataPtr).secondaryContextBase
endif
endcase

'slide time' : begin
(*dataPtr).timeSliderValue = event.value
*(*dataPtr).analyzerPtr->SetTime, event.value

ltgui displayFrame, dataPtr
endcase

'1xname droplist' : begin
mode = WIDGET_INFO(event.id, /droplist_select)
```

```
widget_control, (*dataPtr).selection1XNumberDroplist, get_value=currentNumberList
widget_control, (*dataPtr).selection1XScaleDroplist, get_value=currentScaleList

newNumberList = *(*dataPtr).analyzerPtr->GetScatterIndexNumber(mode)
newScalingList = *(*dataPtr).analyzerPtr->GetScatterIndexScaling(mode)

if(newNumberList[0] ne currentNumberList[0]) then widget_control,
(*dataPtr).selection1XNumberDroplist, set_value = newNumberList
if(newScalingList[0] ne currentScaleList[0]) then widget_control,
(*dataPtr).selection1XScaleDroplist, set_value = newScalingList

ltgui_setScatterMode, 'x', 1, dataPtr

ltgui_displayFrame, dataPtr
endcase

'1yname droplist' : begin
mode = WIDGET_INFO(event.id, /droplist_select)
widget_control, (*dataPtr).selection1YNumberDroplist, get_value=currentNumberList
widget_control, (*dataPtr).selection1YScaleDroplist, get_value=currentScaleList

newNumberList = *(*dataPtr).analyzerPtr->GetScatterIndexNumber(mode)
newScalingList = *(*dataPtr).analyzerPtr->GetScatterIndexScaling(mode)

if(newNumberList[0] ne currentNumberList[0]) then widget_control,
(*dataPtr).selection1YNumberDroplist, set value = newNumberList
if(newScalingList[0] ne currentScaleList[0]) then widget_control,
(*dataPtr).selection1YScaleDroplist, set_value = newScalingList

ltgui_setScatterMode, 'y', 1, dataPtr

ltgui displayFrame, dataPtr
endcase

'1xnumber droplist' : begin
ltgui_setScatterMode, 'x', 1, dataPtr
ltgui_displayFrame, dataPtr
endcase
'1xscale droplist' : begin
ltgui setScatterMode, 'x', 1, dataPtr
ltgui_displayFrame, dataPtr
endcase
'1ynumber droplist' : begin
ltgui setScatterMode, 'y', 1, dataPtr
ltgui_displayFrame, dataPtr
endcase
'1yscale droplist' : begin
ltgui_setScatterMode, 'y', 1, dataPtr
ltgui displayFrame, dataPtr
endcase
'2xname droplist' : begin
mode = WIDGET_INFO(event.id, /droplist_select)
widget_control, (*dataPtr).selection2XNumberDroplist, get_value=currentNumberList
widget_control, (*dataPtr).selection2XScaleDroplist, get_value=currentScaleList

newNumberList = *(*dataPtr).analyzerPtr->GetScatterIndexNumber(mode)
newScalingList = *(*dataPtr).analyzerPtr->GetScatterIndexScaling(mode)

if(newNumberList[0] ne currentNumberList[0]) then widget_control,
(*dataPtr).selection2XNumberDroplist, set value = newNumberList
if(newScalingList[0] ne currentScaleList[0]) then widget_control,
(*dataPtr).selection2XScaleDroplist, set_value = newScalingList

ltgui_setScatterMode, 'x', 2, dataPtr

ltgui displayFrame, dataPtr
endcase

'2yname droplist' : begin
mode = WIDGET_INFO(event.id, /droplist select)
widget control, (*dataPtr).selection2YNumberDroplist, get value=currentNumberList
widget_control, (*dataPtr).selection2YScaleDroplist, get_value=currentScaleList

newNumberList = *(*dataPtr).analyzerPtr->GetScatterIndexNumber(mode)
newScalingList = *(*dataPtr).analyzerPtr->GetScatterIndexScaling(mode)
```

```
if(newNumberList[0] ne currentNumberList[0]) then widget_control,
(*dataPtr).selection2YNumberDroplist, set_value = newNumberList
if(newScalingList[0] ne currentScaleList[0]) then widget_control,
(*dataPtr).selection2YScaleDroplist, set_value = newScalingList

ltgui setScatterMode, 'y', 2, dataPtr
ltgui_displayFrame, dataPtr
endcase

'2xnumber droplist' : begin
ltgui setScatterMode, 'x', 2, dataPtr
ltgui_displayFrame, dataPtr
endcase
'2xscale droplist' : begin
ltgui setScatterMode, 'x', 2, dataPtr
ltgui displayFrame, dataPtr
endcase
'2ynumber droplist' : begin
ltgui_setScatterMode, 'y', 2, dataPtr
ltgui displayFrame, dataPtr
endcase
'2yscale droplist' : begin
ltgui_setScatterMode, 'y', 2, dataPtr
ltgui_displayFrame, dataPtr
endcase
endcase

END

PRO ltgui_setScatterMode, axis, channel, dataPtr
if(channel eq 1) then begin
if(axis eq 'x') then begin
scatterMode = [WIDGET INFO((*dataPtr).selection1XNameDroplist, /droplist_select),
WIDGET INFO((*dataPtr).selection1XNumberDroplist, /droplist select),
WIDGET INFO((*dataPtr).selection1XScaleDroplist, /droplist_select)]
*(*dataPtr).analyzerPtr->SetXScatterMode, 1, scatterMode
endif
if( eq 'y') then begin
scatterMode = [WIDGET INFO((*dataPtr).selection1YNameDroplist, /droplist_select),
WIDGET INFO((*dataPtr).selection1YNumberDroplist, /droplist select),
WIDGET INFO((*dataPtr).selection1YScaleDroplist, /droplist_select)]
*(*dataPtr).analyzerPtr->SetYScatterMode, 1, scatterMode
endif
endif
if(channel eq 2) then begin
if(axis eq 'x') then begin
scatterMode = [WIDGET_INFO((*dataPtr).selection2XNameDroplist, /droplist_select),
WIDGET INFO((*dataPtr).selection2XNumberDroplist, /droplist_select),
WIDGET INFO((*dataPtr).selection2XScaleDroplist, /droplist_select)]
*(*dataPtr).analyzerPtr->SetXScatterMode, 2, scatterMode
endif
if(axis eq 'y') then begin
scatterMode = [WIDGET_INFO((*dataPtr).selection2YNameDroplist, /droplist_select),
WIDGET INFO((*dataPtr).selection2YNumberDroplist, /droplist_select),
WIDGET INFO((*dataPtr).selection2YScaleDroplist, /droplist_select)]
*(*dataPtr).analyzerPtr->SetYScatterMode, 2, scatterMode
endif
endif
END

PRO ltgui_contextBarEvent, event
widget control, event.top, get uvalue=dataPtr
widget_control, event.id, get_uvalue=widget
*(*dataPtr).analyzerPtr->SetFrameMode, widget
ltgui_displayFrame, dataPtr
END


PRO ltgui_loadTrace, EVENT
widget_control, EVENT.top, get_uvalue=dataPtr

if((*dataPtr).currentPath eq '') then begin
newPath = 'D:\tracefiles'
dataFileArray = DIALOG_PICKFILE(filter='*.trc', get_path = newPath, /multiple_files, /READ,
TITLE = 'Load trc file')
(*dataPtr).currentPath = newPath
```

```
endif else begin
newPath = ''
dataFileArray = DIALOG_PICKFILE(filter='*.trc', get_path = newPath, path =
(*dataPtr).currentPath, /multiple_files, /READ, TITLE = 'Load trc file')
(*dataPtr).currentPath = newPath
endelse

sortedData = dataFileArray[sort(dataFileArray)]
if((size(sortedData, /dimensions))[0] gt 0) then begin
for index = 0, (size(sortedData, /dimensions))[0] -1 do begin
if(sortedData[index] ne '') then begin
*(*dataPtr).analyzerPtr->AddTrace, sortedData[index], (*dataPtr).timeSliderValue
endif
endfor
endif
widget_control, (*dataPtr).selection1XNameDroplist, set_value = (*(*dataPtr).analyzerPtr)-
>GetScatterIndexNames()
widget_control, (*dataPtr).selection1YNameDroplist, set_value = (*(*dataPtr).analyzerPtr)-
>GetScatterIndexNames()
widget_control, (*dataPtr).selection2XNameDroplist, set_value = (*(*dataPtr).analyzerPtr)-
>GetScatterIndexNames()
widget_control, (*dataPtr).selection2YNameDroplist, set_value = (*(*dataPtr).analyzerPtr)-
>GetScatterIndexNames()
ltgui_displayFrame, dataPtr

END

PRO ltgui_saveImage, EVENT
widget_control, EVENT.top, get_uvalue=dataPtr
widget_control, event.id, get_uvalue=widget

filename = DIALOG_PICKFILE(default_extension = 'jpg', filter='*.jpg', get_path = newpath, path
= (*dataPtr).currentImagePath, /WRITE, TITLE = 'Save image')
(*dataPtr).currentImagePath = newpath
if(filename eq '') then return
case widget of
'vp1' : begin
wset, (*dataPtr).primaryWinid
write_jpeg, filename, (tvrd(true=1))[*, 0:599, 0:599], quality=100, true=1
endcase
'vp2' : begin
wset, (*dataPtr).secondaryWinid
write_jpeg, filename, (tvrd(true=1))[*, 0:599, 0:599], quality=100, true=1
endcase
endcase
END

PRO ltgui_invertBackground, EVENT
widget_control, EVENT.top, get_uvalue=dataPtr
*(*dataPtr).analyzerPtr->InvertBackground
ltgui_displayFrame, dataPtr
END

PRO ltgui_setClassMode, EVENT
widget_control, EVENT.top, get_uvalue=dataPtr
widget_control, event.id, get_uvalue=widget
*(*dataPtr).analyzerPtr->SetClassMode, widget
ltgui_displayFrame, dataPtr
END

PRO ltgui_traceBreak, EVENT
widget_control, EVENT.top, get_uvalue=dataPtr
stop
;(*(*dataPtr).traceDataArray[0])->LVA_Break
END

PRO ltgui_clearAll, EVENT
widget_control, EVENT.top, get_uvalue=dataPtr
*(*dataPtr).analyzerPtr->ClearTraceBuffer
END

PRO ltgui_scatterPlot, dataPtr, mode
class1arr = ltgui_getResultArray(dataPtr, 1)
class2arr = ltgui_getResultArray(dataPtr, 2)

if(n_elements(class1arr) gt 1) then begin
```

```
if(n_elements(class2arr) gt 1) then begin
ltgui_scatterPlotArray, class1arr, mode, secondClass = class2arr, invertBackground =
(*dataPtr).invertBackground
endif else begin
ltgui scatterPlotArray, class1arr, mode, invertBackground = (*dataPtr).invertBackground
endelse
endif else if(n_elements(class2arr) gt 1) then begin
ltgui_scatterPlotArray, class2arr, mode, invertBackground = (*dataPtr).invertBackground
endif

END

;The display function
PRO ltgui_displayFrame, dataPtr

if(ptr_valid(dataPtr) eq 0) then return

;Display nr 1
wset, (*dataPtr).renderWinid ;Write to render buffer
*(*dataPtr).analyzerPtr->DrawPrimaryFrame

wset, (*dataPtr).primaryWinid ;Copy to frame buffer
Device, Copy= [0,0,600,600,0,0, (*dataPtr).renderWinid]

;Display nr 2
wset, (*dataPtr).renderWinid ;Write to render buffer
*(*dataPtr).analyzerPtr->DrawSecondaryFrame

wset, (*dataPtr).secondaryWinid ;Copy to frame buffer
Device, Copy= [0,0,600,600,0,0, (*dataPtr).renderWinid]
END

PRO ltgui_writeFramebuffer, event
widget_control, EVENT.top, get_uvalue=dataPtr
WRITE_IMAGE,'testfile.bmp', 'bmp', TVRD(TRUE = 1)
END

;I have had enough
PRO ltgui_exit, EVENT
widget_control, event.top, /destroy
END

;The destructor
PRO ltgui_cleanup, ID
widget_control, id, get_uvalue=dataPtr
OBJ_DESTROY, *(*dataPtr).analyzerPtr
ptr_free, (*dataPtr).analyzerPtr
if(ptr_valid(dataPtr) eq 1) then ptr_free, dataPtr
END

;The main procedure
PRO ltgui

DEVICE, DECOMPOSED = 1
;LOADCT, 0

myBase = widget_base(column=1, title = 'Trc reader', MBAR = fileBar, /CONTEXT_EVENTS)

file_menu      = widget_button(fileBar, value='File', /menu)
file_bttnOpen  = widget_button(file_menu, value='Load trc data', event_pro='ltgui_loadTrace')
file_bttnOpen  = widget_button(file_menu, value='Save viewport 1', uvalue = 'vp1',
event_pro='ltgui_saveImage')
file_bttnOpen  = widget_button(file_menu, value='Save viewport 2', uvalue = 'vp2',
event_pro='ltgui_saveImage')
file_bttnExit  = widget_button(file_menu, value='Exit', event_pro='ltgui_exit')

tool_menu          = widget_button(fileBar, value='Tools', /menu)
tool_bttnInvback = widget_button(tool_menu, value='Invert background', event_pro =
'ltgui_invertBackground')
tool_classMode  = widget_button(tool_menu, value='Class mode', /menu)
classMode_normal =widget_button(tool_classMode, value='Normal', uvalue='Normal', event_pro =
'ltgui_setClassMode')
classMode_cacer =widget_button(tool_classMode, value='Liposarcoma', uvalue = 'Liposarcoma',
event_pro = 'ltgui_setClassMode')
classMode_age =widget_button(tool_classMode, value='Age', uvalue = 'Age', event_pro =
'ltgui_setClassMode')
```

```
classMode_size =widget_button(tool_classMode, value='Tumor size', uvalue ='Tumor size',
event_pro = 'ltgui_setClassMode')
classMode size =widget button(tool_classMode, value='Trunkus', uvalue ='Tumor position',
event_pro = 'ltgui_setClassMode')

tool bttnBreak  = widget button(tool menu, value='Break', event pro = 'ltgui traceBreak')
tool_bttnClearAll = widget button(tool_menu, value='Clear all', event_pro = 'ltgui_clearAll')
;tool_bttnWrite = widget_button(tool_menu, value='Write framebuffer',
event_pro='ltgui_writeFramebuffer')

;The output window
windowBase     = widget base(mybase, /ROW, /align_center)
primaryWindow   = widget draw(windowBase, xsize = 600, ysize = 600, uvalue = 'primaryWindow',
keyboard_events = 1, /BUTTON_EVENTS)
secondaryWindow     = widget draw(windowBase, xsize = 600, ysize = 600, uvalue =
'secondaryWindow', keyboard_events = 1, /BUTTON_EVENTS)

Window, /PIXMAP, XSIZE = 600, YSIZE = 600, /FREE
renderWindow = !D.Window

;Second row
;buttonBase = widget_base(myBase, uvalue='Display plasma', /ROW, /align_center)
;buttonSize = 65

;Output window context base
primaryContextBase      = widget base(myBase, /context menu)
k1Button                = widget button(primaryContextBase, value='k1 histogram',
uvalue='k1 primary',        event pro='ltgui contextBarEvent')
k2Button                = widget button(primaryContextBase, value='k2 histogram',
uvalue='k2 primary',        event_pro='ltgui_contextBarEvent')
k3Button                = widget button(primaryContextBase, value='k3 histogram',
uvalue='k3 primary',        event pro='ltgui contextBarEvent')
r2Button                = widget button(primaryContextBase, value='r2 histogram',
uvalue='r2 primary',        event pro='ltgui contextBarEvent')
plasmaButton            = widget button(primaryContextBase, value='Plasma',
uvalue='plasma primary',    event_pro='ltgui_contextBarEvent')
tumorButton             = widget button(primaryContextBase, value='Tumor Histogram',
uvalue='tumor primary',     event pro='ltgui contextBarEvent')
ratioButton             = widget button(primaryContextBase, value='Tumor tissue ratio',
uvalue='ttratio primary',   event pro='ltgui contextBarEvent')
slopeButton             = widget button(primaryContextBase, value='Patlak slope',
uvalue='slope primary',     event_pro='ltgui_contextBarEvent')
interceptButton         = widget button(primaryContextBase, value='Patlak intercept',
uvalue='intercept primary', event pro='ltgui contextBarEvent')
metabolicButton         = widget button(primaryContextBase, value='MB histogram',
uvalue='metabolic primary', event_pro='ltgui_contextBarEvent')
vascularButton         = widget button(primaryContextBase, value='VB histogram',
uvalue='vb primary', event_pro='ltgui_contextBarEvent')
legendButton            = widget button(primaryContextBase, value='Legend',
uvalue='legend primary',    event pro='ltgui contextBarEvent')
legendButton            = widget button(primaryContextBase, value='U matrix',
uvalue='umatrix primary',    event_pro='ltgui_contextBarEvent')
t90pButton              = widget button(primaryContextBase, value='Tumor 95p',
uvalue='tum90p primary',    event_pro='ltgui_contextBarEvent')
t90pScaledButton        = widget button(primaryContextBase, value='Tumor 95p scaled',
uvalue='tum90p scaled primary',    event pro='ltgui contextBarEvent')
tpeakButton             = widget button(primaryContextBase, value='Tumor peak',
uvalue='tumpeak primary',   event_pro='ltgui_contextBarEvent')
tpeakScaledButton       = widget button(primaryContextBase, value='Tumor peak scaled',
uvalue='tumpeak scaled primary',   event pro='ltgui contextBarEvent')
tmaxButton              = widget button(primaryContextBase, value='Tumor max',
uvalue='tummax primary',    event pro='ltgui contextBarEvent')
tmaxScaledButton        = widget button(primaryContextBase, value='Tumor max scaled',
uvalue='tummax scaled primary',    event_pro='ltgui_contextBarEvent')
tmedianButton           = widget button(primaryContextBase, value='Tumor Mean',
uvalue='tummean primary',   event pro='ltgui contextBarEvent')
tmedianScaledButton     = widget button(primaryContextBase, value='Tumor Mean scaled',
uvalue='tummean scaled primary',   event_pro='ltgui_contextBarEvent')
tvarianceButton         = widget button(primaryContextBase, value='Tumor Variance',
uvalue='tumvariance primary', event pro='ltgui contextBarEvent')
tskewButton             = widget button(primaryContextBase, value='Tumor Skew',
uvalue='tumskew primary',   event pro='ltgui contextBarEvent')
tkurtosisButton         = widget button(primaryContextBase, value='Tumor Kurtosis',
uvalue='tumkurtos primary', event_pro='ltgui_contextBarEvent')
scatterButton           = widget button(primaryContextBase, value='Scatter plot 1',
uvalue='scatter 1 primary',       event_pro='ltgui_contextBarEvent')
```

```
scatterButton           = widget button(primaryContextBase, value='Scatter plot 2',
uvalue='scatter 2 primary',       event_pro='ltgui_contextBarEvent')
sequenceButton          = widget button(primaryContextBase, value='SUVe98 sequence',
uvalue='sequence SUVe98 primary',       event pro='ltgui contextBarEvent')
sequenceButton          = widget button(primaryContextBase, value='SUVe98r sequence',
uvalue='sequence SUVe98r primary',       event pro='ltgui contextBarEvent')
sequenceButton          = widget button(primaryContextBase, value='SUVl98 sequence',
uvalue='sequence SUVl98 primary',       event_pro='ltgui_contextBarEvent')
sequenceButton          = widget button(primaryContextBase, value='SUVl98r sequence',
uvalue='sequence SUVl98r primary',       event pro='ltgui contextBarEvent')
sequenceButton          = widget button(primaryContextBase, value='K1 sequence',
uvalue='sequence K1 primary',       event_pro='ltgui_contextBarEvent')
sequenceButton          = widget button(primaryContextBase, value='Metabolic sequence',
uvalue='sequence MET primary',       event_pro='ltgui_contextBarEvent')
sequenceButton          = widget button(primaryContextBase, value='VB sequence',
uvalue='sequence VB primary',       event pro='ltgui contextBarEvent')
sequenceButton          = widget button(primaryContextBase, value='EM special sequence',
uvalue='sequence HET primary',       event_pro='ltgui_contextBarEvent')
correlationButton          = widget button(primaryContextBase, value='Correlation matrix',
uvalue='correlationMatrix primary',       event_pro='ltgui_contextBarEvent')

secondaryContextBase    = widget base(myBase, /context menu)
k1Button                = widget button(secondaryContextBase, value='k1 histogram',
uvalue='k1 secondary',         event_pro='ltgui_contextBarEvent')
k2Button                = widget button(secondaryContextBase, value='k2 histogram',
uvalue='k2 secondary',         event pro='ltgui contextBarEvent')
k3Button                = widget button(secondaryContextBase, value='k3 histogram',
uvalue='k3 secondary',         event pro='ltgui contextBarEvent')
r2Button                = widget button(secondaryContextBase, value='r2 histogram',
uvalue='r2 secondary',         event_pro='ltgui_contextBarEvent')
plasmaButton            = widget button(secondaryContextBase, value='Plasma',
uvalue='plasma secondary',     event pro='ltgui contextBarEvent')
tumorButton             = widget button(secondaryContextBase, value='Tumor Histogram',
uvalue='tumor secondary',      event pro='ltgui contextBarEvent')
ratioButton             = widget button(secondaryContextBase, value='Tumor tissue ratio',
uvalue='ttratio secondary',    event_pro='ltgui_contextBarEvent')
slopeButton             = widget button(secondaryContextBase, value='Patlak slope',
uvalue='slope secondary',      event pro='ltgui contextBarEvent')
interceptButton         = widget button(secondaryContextBase, value='Patlak intercept',
uvalue='intercept secondary', event pro='ltgui contextBarEvent')
metabolicButton         = widget button(secondaryContextBase, value='metabolic rate',
uvalue='metabolic secondary', event_pro='ltgui_contextBarEvent')
vascularButton          = widget button(secondaryContextBase, value='VB histogram',
uvalue='vb secondary', event pro='ltgui contextBarEvent')
legendButton            = widget button(secondaryContextBase, value='Legend',
uvalue='legend secondary',     event_pro='ltgui_contextBarEvent')
legendButton            = widget button(secondaryContextBase, value='U matrix',
uvalue='umatrix secondary',    event_pro='ltgui_contextBarEvent')
t90pButton              = widget button(secondaryContextBase, value='Tumor 98p',
uvalue='tum90p secondary',     event pro='ltgui contextBarEvent')
t90pScaledButton        = widget button(secondaryContextBase, value='Tumor 98p scaled',
uvalue='tum90p scaled secondary',    event_pro='ltgui_contextBarEvent')
tpeakButton             = widget button(secondaryContextBase, value='Tumor peak',
uvalue='tumpeak secondary',    event_pro='ltgui_contextBarEvent')
tpeakScaledButton       = widget button(secondaryContextBase, value='Tumor scaled peak',
uvalue='tumpeak scaled secondary',    event pro='ltgui contextBarEvent')
tmaxButton              = widget button(secondaryContextBase, value='Tumor max',
uvalue='tummax secondary',     event_pro='ltgui_contextBarEvent')
tmaxScaledButton        = widget button(secondaryContextBase, value='Tumor max scaled',
uvalue='tummax scaled secondary',    event pro='ltgui contextBarEvent')
medianButton            = widget button(secondaryContextBase, value='Tumor Mean',
uvalue='tummean secondary',   event pro='ltgui contextBarEvent')
medianScaledButton      = widget button(secondaryContextBase, value='Tumor Mean scaled',
uvalue='tummean scaled secondary',   event_pro='ltgui_contextBarEvent')
varianceButton          = widget button(secondaryContextBase, value='Tumor Variance',
uvalue='tumvariance secondary', event pro='ltgui contextBarEvent')
tskewButton             = widget button(secondaryContextBase, value='Tumor Skew',
uvalue='tumskew secondary',          event_pro='ltgui_contextBarEvent')
tkurtosisButton         = widget button(secondaryContextBase, value='Tumor Kurtosis',
uvalue='tumkurtos secondary',       event pro='ltgui contextBarEvent')
scatterButton           = widget button(secondaryContextBase, value='Scatter 1',
uvalue='scatter 1 secondary',       event pro='ltgui contextBarEvent')
scatterButton           = widget button(secondaryContextBase, value='Scatter 2',
uvalue='scatter 2 secondary',       event_pro='ltgui_contextBarEvent')
sequenceButton          = widget button(secondaryContextBase, value='SUVe98 sequence',
uvalue='sequence SUVe98 secondary',       event_pro='ltgui_contextBarEvent')
```

```
sequenceButton          = widget_button(secondaryContextBase, value='SUVe98r sequence',
uvalue='sequence SUVe98r secondary',       event_pro='ltgui_contextBarEvent')
sequenceButton          = widget_button(secondaryContextBase, value='SUVl98 sequence',
uvalue='sequence SUVl98 secondary',       event pro='ltgui contextBarEvent')
sequenceButton          = widget button(secondaryContextBase, value='SUVl98r sequence',
uvalue='sequence SUVl98r secondary',       event pro='ltgui contextBarEvent')
sequenceButton          = widget_button(secondaryContextBase, value='K1 sequence',
uvalue='sequence K1 secondary',       event_pro='ltgui_contextBarEvent')
sequenceButton          = widget_button(secondaryContextBase, value='Metabolic sequence',
uvalue='sequence MET secondary',       event pro='ltgui contextBarEvent')
sequenceButton          = widget button(secondaryContextBase, value='VB sequence',
uvalue='sequence VB secondary',       event_pro='ltgui_contextBarEvent')
sequenceButton          = widget_button(secondaryContextBase, value='EM special sequence',
uvalue='sequence HET secondary',       event_pro='ltgui_contextBarEvent')
correlationButton          = widget_button(secondaryContextBase, value='Correlation matrix',
uvalue='correlationMatrix secondary',       event_pro='ltgui_contextBarEvent')

sliderbase              = widget_base(myBase, /ROW, /align_center)
depthSelectSlider       = widget_slider(sliderbase, MAXIMUM = 50, MINIMUM = 0, uvalue='slide
time')

tempAnalyzerPtr = ptr_new(OBJ_NEW('LVA_TraceAnalyzer'))

scatterbase = widget_base(myBase, /COLUMN, /align_center)
*tempAnalyzerPtr->LVA_Constructor

xSelectorArray  = *tempAnalyzerPtr->GetScatterIndexNames()
xNumberArray    = *tempAnalyzerPtr->GetScatterIndexNumber(0)
xScalingArray   = *tempAnalyzerPtr->GetScatterIndexScaling(0)
ySelectorArray  = *tempAnalyzerPtr->GetScatterIndexNames()
yNumberArray    = *tempAnalyzerPtr->GetScatterIndexNumber(0)
yScalingArray   = *tempAnalyzerPtr->GetScatterIndexScaling(0)

scatter1base              = widget base(scatterbase, /ROW, /align center)
scatter1label   = Widget_Label(scatter1base, Value='   First scatterplot:    ')
selection1XNameDroplist = widget_droplist(scatter1base, value = xSelectorArray, uvalue='1xname
droplist')
selection1XNumberDroplist = widget_droplist(scatter1base, value = xNumberArray,
uvalue='1xnumber droplist')
selection1XScaleDroplist = widget_droplist(scatter1base, value = xScalingArray,
uvalue='1xscale droplist')
selection1YNameDroplist = widget_droplist(scatter1base, value = ySelectorArray, uvalue='1yname
droplist')
selection1YNumberDroplist = widget_droplist(scatter1base, value = yNumberArray,
uvalue='1ynumber droplist')
selection1YScaleDroplist = widget_droplist(scatter1base, value = yScalingArray,
uvalue='1yscale droplist')


xSelectorArray  = *tempAnalyzerPtr->GetScatterIndexNames()
xNumberArray    = *tempAnalyzerPtr->GetScatterIndexNumber(0)
xScalingArray   = *tempAnalyzerPtr->GetScatterIndexScaling(0)
ySelectorArray  = *tempAnalyzerPtr->GetScatterIndexNames()
yNumberArray    = *tempAnalyzerPtr->GetScatterIndexNumber(0)
yScalingArray   = *tempAnalyzerPtr->GetScatterIndexScaling(0)

scatter2base              = widget_base(scatterbase, /ROW, /align_center)
scatter2label   = Widget_Label(scatter2base, Value='Second scatterplot: ')
selection2XNameDroplist = widget_droplist(scatter2base, value = xSelectorArray, uvalue='2xname
droplist')
selection2XNumberDroplist = widget_droplist(scatter2base, value = xNumberArray,
uvalue='2xnumber droplist')
selection2XScaleDroplist = widget_droplist(scatter2base, value = xScalingArray,
uvalue='2xscale droplist')
selection2YNameDroplist = widget_droplist(scatter2base, value = ySelectorArray, uvalue='2yname
droplist')
selection2YNumberDroplist = widget_droplist(scatter2base, value = yNumberArray,
uvalue='2ynumber droplist')
selection2YScaleDroplist = widget_droplist(scatter2base, value = yScalingArray,
uvalue='2yscale droplist')

;Activate the hierarchy
widget_control, myBase, /realize

widget_control, primaryWindow, get_value=primaryWinid
widget_control, secondaryWindow, get_value=secondaryWinid
```

```
; Our data struct
ltguiData = {$
renderWinid:renderWindow,$
primaryWinid:primaryWinid,$
secondaryWinid:secondaryWinid,$
primaryContextBase:primaryContextBase,$
secondaryContextBase:secondaryContextBase,$
timeSliderValue:0.0,$
currentPath:'D:\tracefiles',$
refresh:0,$
invertBackground:0L,$
xScatterIndex:0L,$
yScatterIndex:0L,$
selection1XNameDroplist:selection1XNameDroplist,$
selection1XNumberDroplist:selection1XNumberDroplist,$
selection1XScaleDroplist:selection1XScaleDroplist,$
selection1YNameDroplist:selection1YNameDroplist,$
selection1YNumberDroplist:selection1YNumberDroplist,$
selection1YScaleDroplist:selection1YScaleDroplist,$
selection2XNameDroplist:selection2XNameDroplist,$
selection2XNumberDroplist:selection2XNumberDroplist,$
selection2XScaleDroplist:selection2XScaleDroplist,$
selection2YNameDroplist:selection2YNameDroplist,$
selection2YNumberDroplist:selection2YNumberDroplist,$
selection2YScaleDroplist:selection2YScaleDroplist,$
analyzerPtr:ptr new(),$
currentImagePath:'D:\outputfiles'$
}
;Our actual storage
dataPtr = ptr_new(ltguiData)

(*dataPtr).analyzerPtr = tempAnalyzerPtr
widget_control, myBase, set_uvalue=dataPtr

; Start the manager
xmanager, 'ltgui', mybase, cleanup = 'ltgui_cleanup', /no_block

;COMMON T, draw
;draw = plotWindow

END
```

Trace analyzer

```
PRO LVA_TraceAnalyzer__define
struct =$
{$
LVA_TraceAnalyzer, $
maxTraceData:0L,$
activeTraceData:0L,$
traceDataArray:ptrarr(20),$
sequencesArray:ptrarr(4,4),$
class1Color:0L,$
class2Color:0L,$
UMatrix:fltarr(4,4,5),$
UColor:fltarr(9),$
invertBackground:0L,$
selectorArray:make array(11, /STRING),$
SUVSubArray:make_array(7, /STRING),$
kSubArray:make_array(5, /STRING),$
tumorSubArray:make_array(9, /STRING),$
scaleSelectorArray:make array(2, /STRING),$
emptySelectorArray:make_array(1, /STRING),$

;The display modes
primaryWinDisplayMode:0L,$
secondaryWinDisplayMode:0L,$

; The histogram ranges
k1maxx:2.5,$
…
scatter2yMode:0L,$
```

```
scatter2ynumber:0L,$
scatter2yscale:0L$
}
END

PRO LVA_TraceAnalyzer::AddTrace, filename, timeValue ;TimeValue is a legacy variable, please
remove
if(self.activeTraceData lt self.maxTraceData) then begin
*self.traceDataArray[self.activeTraceData]->LVA_LoadTrace, filename, timeValue
self.activeTraceData++
endif
self->SetClassMode, 'Liposarcoma'
END

PRO LVA_TraceAnalyzer::ClearTraceBuffer
self.activeTraceData = 0
END

PRO LVA_TraceAnalyzer::InvertBackground
for tDataNr = 0, self.activeTraceData -1 do begin
*self.traceDataArray[tDataNr]->LVA_ToggleInvertBackground
endfor
if(self.invertBackground eq 0) then self.invertBackground = 1 else self.invertBackground = 0
END

PRO LVA_TraceAnalyzer::SetTime, time
for index = 0, self.maxTraceData -1 do begin
*self.traceDataArray[index]->LVA_SetTime, time
endfor
END

PRO LVA_TraceAnalyzer::SetFrameMode, widget
case widget of
'k1 primary'            : self.primaryWinDisplayMode = 1
'k2 primary'            : self.primaryWinDisplayMode = 2
…
'correlationMatrix secondary' : self.secondaryWinDisplaymode = 41

else: print, 'unrecognized widget: ', widget
endcase
END

PRO LVA_TraceAnalyzer::SetClassMode, widget
classMode = 0
case widget of
'Normal'        : classmode  = 0
'Liposarcoma'   : classmode  = 1
'Age'           : classmode  = 2
'Tumor size'    : classmode  = 3
'Tumor position': classmode  = 4
else            : print, 'Unhandled widget in ltgui_setClassMode'
endcase

for tDataNr = 0, self.activeTraceData -1 do begin
*self.traceDataArray[tDataNr]->LVA_SetClassMode, classmode
endfor
END

FUNCTION LVA_TraceAnalyzer::GetULvl, n1, n2, psig
if(n2 lt n1) then begin
n1ind = n1 - 6
n2ind = n2 - 2
endif else begin
n1ind = n2 -6
n2ind = n1 -2
endelse
case psig of
0.2 : pind = 0
0.1 : pind = 1
0.05 : pind = 2
0.02 : pind = 3
0.01 : pind = 4
else : pind = -1
endcase

if(pind ge 0)then return, self.UMatrix[n1ind, n2ind, pind] else return, -1
```

```
END

FUNCTION LVA_TraceAnalyzer::GetCorrColor, corVal
color = self.UColor[4]
if(corVal ge 0.39) then color = self.UColor[3]
if(corVal ge 0.5) then color = self.UColor[1]
if(corVal ge 0.6) then color = self.UColor[0]

return, color
END

FUNCTION LVA_TraceAnalyzer::GetUColor, n1, n2, uVal
uMax = n1*n2

color = self.UColor[4]
toplimit = self->GetULvl(n1, n2, 0.02)
hightlimit = self->GetULvl(n1, n2, 0.05)
if(uVal ge self->GetULvl(n1, n2, 0.2)-0.1) then color = self.UColor[5]
if(uVal ge self->GetULvl(n1, n2, 0.1)-0.1) then color = self.UColor[6]
if(hightlimit ne -1 AND uVal ge hightlimit-0.1) then color = self.UColor[7]
if(toplimit ne -1 AND uVal ge toplimit -0.1) then color = self.UColor[8]

if(uVal le uMax - self->GetULvl(n1, n2, 0.2)+0.1) then color = self.UColor[3]
if(uVal le uMax - self->GetULvl(n1, n2, 0.1)+0.1) then color = self.UColor[2]
if(hightlimit ne -1 AND uVal le uMax - hightlimit+0.1) then color = self.UColor[1]
if(toplimit ne -1 AND uVal le uMax - toplimit+0.1) then color = self.UColor[0]

return, color
END

FUNCTION LVA_TraceAnalyzer::UValueSignificant, n1, n2, uval
uMax = n1*n2
if(uVal le uMax -uVal) then uMir = uMax -uVal else uMir = uVal
if(uMir ge self->GetULvl(n1, n2, 0.05)) then return, 1 else return, 0
END

PRO LVA_TraceAnalyzer::SetXScatterMode, id, mode
if(n_elements(mode) eq 3) then begin
if(id eq 1) then begin
self.scatter1xMode = mode[0]
self.scatter1xnumber = mode[1]
self.scatter1xscale = mode[2]
endif
if(id eq 2) then begin
self.scatter2xMode = mode[0]
self.scatter2xnumber = mode[1]
self.scatter2xscale = mode[2]
endif
endif
END

PRO LVA_TraceAnalyzer::SetYScatterMode, id, mode
if(n_elements(mode) eq 3) then begin
if(id eq 1) then begin
self.scatter1yMode = mode[0]
self.scatter1ynumber = mode[1]
self.scatter1yscale = mode[2]
endif
if(id eq 2) then begin
self.scatter2yMode = mode[0]
self.scatter2ynumber = mode[1]
self.scatter2yscale = mode[2]
endif
endif
END

PRO LVA_TraceAnalyzer::DrawPrimaryFrame
self->DrawFrame, self.primaryWinDisplayMode
END

PRO LVA_TraceAnalyzer::DrawSecondaryFrame
self->DrawFrame, self.secondaryWinDisplayMode
END

PRO LVA_TraceAnalyzer::DrawFrame, drawmode
if(self.activeTraceData eq 0) then return
```

```
for tDataNr = 0, self.activeTraceData -1 do begin
case drawmode of
1   :   *self.traceDataArray[tDataNr]->PlotK1, self.k1maxx, self.k1maxy, tDataNr
2   :   *self.traceDataArray[tDataNr]->PlotK2, self.k2maxx, self.k2maxy, tDataNr
3   :   *self.traceDataArray[tDataNr]->PlotK3, self.k3maxx, self.k3maxy, tDataNr
4   :   *self.traceDataArray[tDataNr]->PlotPlasma, self.plasmamaxx, self.plasmamaxy, tDataNr
5   :   *self.traceDataArray[tDataNr]->PlotTumor, self.tumormaxx, self.tumormaxy, tDataNr
6   :   *self.traceDataArray[tDataNr]->PlotTumorRatio, self.ratiomaxx, self.ratiomaxy, tDataNr
7   :   *self.traceDataArray[tDataNr]->plotPatlakSlopeHistogram
8   :   *self.traceDataArray[tDataNr]->plotPatlakInterceptHistogram
9   :   *self.traceDataArray[tDataNr]->plotMBHistogram
10  :   begin
if(tDataNr eq 0 ) then *self.traceDataArray[tDataNr]->PrintLegendHeader, 10 , 580
*self.traceDataArray[tDataNr]->PrintLegend, 10, 560-20*tDataNr, 0
endcase
11  :   *self.traceDataArray[tDataNr]->PlotTumor90p, self.t90pmaxx, self.t90pmaxy, tDataNr
12  :   *self.traceDataArray[tDataNr]->PlotTumorMean, self.meanmaxx, self.meanmaxy, tDataNr
13  :   *self.traceDataArray[tDataNr]->PlotTumorVariance, self.varimaxx, self.varimaxy,
tDataNr
14  :   *self.traceDataArray[tDataNr]->PlotTumorSkew, self.skewmaxx, self.skewmaxy, tDataNr
15  :   *self.traceDataArray[tDataNr]->PlotTumorKurtosis, self.kurtmaxx, self.kurtmaxy,
tDataNr
16  :   *self.traceDataArray[tDataNr]->PlotTumorPeak, self.peakmaxx, self.peakmaxy, tDataNr
17  :   *self.traceDataArray[tDataNr]->PlotTumorMax, self.maxmaxx, self.maxmaxy, tDataNr
18  :   *self.traceDataArray[tDataNr]->PlotR2, self.pearsonmaxx, self.pearsonmaxy, tDataNr
19  :   if(tDataNr eq 0 ) then self->PlotUmatrix, 30, 500
20  :   *self.traceDataArray[tDataNr]->PlotPatlak, 's', self.patlakslopemaxx,
self.patlakslopemaxy, tDataNr
21  :   *self.traceDataArray[tDataNr]->PlotPatlak, 'i', self.patlakinterceptmaxx,
self.patlakinterceptmaxy, tDataNr
22  :   *self.traceDataArray[tDataNr]->plotVBHistogram
27  :   *self.traceDataArray[tDataNr]->PlotTumorMaxScaled, self.maxmaxx, self.maxmaxy, tDataNr
28  :   *self.traceDataArray[tDataNr]->PlotTumorPeakScaled, self.maxmaxx, self.peakmaxy,
tDataNr
29  :   *self.traceDataArray[tDataNr]->PlotTumor90pScaled, self.maxmaxx, self.t90pmaxy,
tDataNr
30  :   *self.traceDataArray[tDataNr]->PlotTumorMeanScaled, self.maxmaxx, self.meanmaxy,
tDataNr
31  :   if(tDataNr eq 0) then self->PlotScatter, 1, self.scatter1xMode, self.scatter1xnumber,
self.scatter1xscale, self.scatter1yMode, self.scatter1ynumber, self.scatter1yscale
32  :   if(tDataNr eq 0) then self->PlotScatter, 2, self.scatter2xMode, self.scatter2xnumber,
self.scatter2xscale, self.scatter2yMode, self.scatter2ynumber, self.scatter2yscale
33  :   if(tDataNr eq 0 ) then self->PlotSequence, 1
34  :   if(tDataNr eq 0 ) then self->PlotSequence, 2
35  :   if(tDataNr eq 0 ) then self->PlotSequence, 3
36  :   if(tDataNr eq 0 ) then self->PlotSequence, 4
37  :   if(tDataNr eq 0 ) then self->PlotSequence, 5
38  :   if(tDataNr eq 0 ) then self->PlotSequence, 6
39  :   if(tDataNr eq 0 ) then self->PlotSequence, 7
40  :   if(tDataNr eq 0 ) then self->PlotSequence, 8
41  :   if(tDataNr eq 0 ) then self->PlotCorrelationmatrix, 30, 500
else: return
endcase
endfor

if(drawmode eq 10) then begin
if(self.invertBackground eq 1) then begin
foregroundColor = 0
backgroundColor = 16777215
endif else begin
foregroundColor = 16777215
backgroundColor = 0
endelse

names = StrArr(self.activeTraceData)
lineStyles = IntArr(self.activeTraceData)
psyms = IntArr(self.activeTraceData)
colors = FltArr(self.activeTraceData)
for tDataNr = 0, self.activeTraceData -1 do begin
names[tDataNr] = *self.traceDataArray[tDataNr]->GetNameStr()
psyms[tDataNr] = -((tDataNr +3) MOD 6) -1
if(psyms[tDataNr] eq -3) then psyms[tDataNr] = -7
lineStyles[tDataNr] = (tDataNr) MOD 5
colors[tDataNr] = *self.traceDataArray[tDataNr]->LVA_getLineColor()
if(colors[tDataNr] eq backgroundColor) then colors[tDataNr] = foregroundColor
```

```
endfor
legend, names,linestyle=lineStyles, psym = psyms, color = colors, textcolors = colors,
charsize = 1.5, position = [0.1, 0.4], pspacing = 1
endif
END

;;;;;;;;;;;;;;;;;;
;; Scatter plots ;;
;;;;;;;;;;;;;;;;;;

FUNCTION LVA_TraceAnalyzer::GetSUVLabel, number
case number of
0 : result = 'Max of '
1 : result = 'Peak of '
2 : result = '98 percentile of '
3 : result = 'Median of '
4 : result = 'Variance of '
5 : result = 'Skewness of '
6 : result = 'Kurtosis of '
else : result = 'GetSUVLabel error :'+string(number)
endcase
return, result
END

FUNCTION LVA_TraceAnalyzer::GetKLabel, number
case number of
0 : result = 'Median of '
1 : result = '90 percentile of '
2 : result = 'Variance of '
3 : result = 'Skewness of '
4 : result = 'Kurtosis of '
else : result = 'GetKLabel error :'+string(number)
endcase
return, result
END

FUNCTION LVA_TraceAnalyzer::GetTumorLabel, number
case number of
0 : result = 'at 2 min'
1 : result = 'at 45 min'
2 : result = 'K1'
3 : result = 'k2'
4 : result = 'k3'
5 : result = 'Metabolic rate'
6 : result = 'Patlak slope'
7 : result = 'Patlak intercept'
8 : result = 'Vascular fraction'
endcase
return, result
END

FUNCTION LVA_TraceAnalyzer::GetLabel, name, number, scaling

if(self.scatterMode eq 1) then begin
case name of
0 : begin
if(scaling eq 1) then typeStr = 'SUV/tissue' else typeStr = 'SUV'
result = self->GetSUVLabel(number) + typeStr + ' at 2 min'
endcase
1 : begin
if(scaling eq 1) then typeStr = 'SUV/tissue' else typeStr = 'SUV'
result = self->GetSUVLabel(number) + typeStr + ' at 45 min'
endcase
2 : begin
if(scaling eq 1) then typeStr = 'SUV/tissue' else typeStr = 'SUV'
result = self->GetSUVLabel(number) + typeStr + ' at 2+45 min'
endcase
3 : result = self->GetKLabel(number) + 'k1'
4 : result = self->GetKLabel(number) + 'k2'
5 : result = self->GetKLabel(number) + 'k3'
6 : result = self->GetKLabel(number) + 'metabolism'
7 : result = self->GetKLabel(number) + 'Patlak slope'
8 : result = self->GetKLabel(number) + 'Patlak intercept'
9 : result = self->GetKLabel(number) + 'r2'
10 : result = self->GetKLabel(number) + 'vb'
else : result = 'GetLabel error'
```

```
endcase
endif
if(self.scatterMode eq 2) then begin
if(scaling eq 1) then typeStr = 'SUV/tissue ' else typeStr = 'SUV '
if(number lt 2) then result = typeStr + self->GetTumorLabel(number) else result = self-
>GetTumorLabel(number)
endif
return, result
END

PRO LVA_TraceAnalyzer::PlotCorrelationmatrix, xpos, ypos
if(self.invertBackground eq 1) then begin
backgroundColor = 16777215
foregroundColor  = 0
endif else begin
backgroundColor = 0
foregroundColor  = 16777215
endelse

clearBuffer = replicate(backgroundColor, 600, 600)
tv, clearBuffer

xspacing = 50
yspacing = 30

nchar = 1.8
numchar = 1.2
name  = 0
number = 0
scaling = 0

nameArr = make_array(9, /STRING)
nameArr[0] = 'SUV!UE'
nameArr[1] = 'SUV!UL'
nameArr[2] = 'K1'
nameArr[3] = 'k2'
nameArr[4] = 'k3'
nameArr[5] = 'vb'
nameArr[6] = 'MR!DPET'
nameArr[7] = 'Pat!Ds'
nameArr[8] = 'Pat!Di'

topx = xpos +50
for ind = 0, 7 do begin
xyouts, topx, ypos, nameArr[8-ind], color = foregroundColor, alignment = 0.5, /DEVICE,
charsize = nchar
topx += xspacing
endfor
ypos -= yspacing

for ind = 0, 7 do begin
linex = xpos
xyouts, linex, ypos, nameArr[ind], color = foregroundColor, alignment = 0.5, /DEVICE, charsize
= nchar
for lind = 0, 7-ind do begin
linex += xspacing
if(ind ne 8-lind) then begin
lcorr = 0
for tDataNr = 0, self.activeTraceData -1 do begin
lcorr += *self.traceDataArray[tDataNr]->LVA_GetCorrelation(ind, 8-lind)
endfor
lcorr /= float(self.activeTraceData)
scorr = string(lcorr, format = '(F5.2)')
boxColor = self->GetCorrColor(lcorr)
xs = 46
ys = 28
xlp = linex -19
ylp = ypos -12
x = [xlp, xlp+xs, xlp+xs, xlp]
y = [ylp, ylp, ylp+ys, ylp+ys]
if(boxColor eq foregroundColor) then boxColor = backgroundColor
POLYFILL, x, y, color = boxColor, /DEVICE
endif else scorr = 'X'
xyouts, linex, ypos, scorr , color = foregroundColor, alignment = 0.5, /DEVICE, charsize =
numchar
endfor
```

```
ypos -= yspacing
endfor

END

PRO LVA_TraceAnalyzer::PlotUmatrix, xpos, ypos
if(self.invertBackground eq 1) then begin
backgroundColor = 16777215
foregroundColor  = 0
endif else begin
backgroundColor = 0
foregroundColor  = 16777215
endelse

clearBuffer = replicate(backgroundColor, 600, 600)
tv, clearBuffer

xspacing = 48
yspacing = 30

nchar = 1.4
name  = 0
number = 0
scaling = 0
topx = xpos +60
xyouts, topx, ypos, 'Peak', color = foregroundColor, alignment = 0.5, /DEVICE,charsize = nchar
topx += xspacing
xyouts, topx, ypos, 'max', color = foregroundColor, alignment = 0.5, /DEVICE, charsize = nchar
topx += xspacing
…
self->PlotUmatrixLine, 8, 0, xpos, ypos, xspacing
ypos -= yspacing
self->PlotUmatrixLine, 9, 0, xpos, ypos, xspacing

for tdat = 0, self.activeTraceData -1 do begin
ypos -= yspacing
self->PlotR2Line, tdat, xpos, ypos, xspacing
endfor


END

PRO LVA_TraceAnalyzer::PlotUmatrixLine, name, scaling, xposref, yposref, xspacing
if(self.invertBackground eq 1) then begin
backgroundColor = 16777215
foregroundColor  = 0
endif else begin
backgroundColor = 0
foregroundColor  = 16777215
endelse

nchar = 1.2
schar = 1.4
tchar = 1.8

xpos = xposref
ypos = yposref

if(scaling eq 1 AND name eq 0) then begin
xyouts, xpos, ypos, 'SUV!S!Dr!R!UL', color = foregroundColor, alignment = 0.5, /DEVICE,
charsize = tchar
endif else begin
if(scaling eq 1 AND name eq 1) then begin
xyouts, xpos, ypos, 'SUV!S!Dr!R!UE', color = foregroundColor, alignment = 0.5, /DEVICE,
charsize = tchar
endif else begin
xyouts, xpos, ypos, self.selectorArray[name], color = foregroundColor, alignment = 0.5,
/DEVICE, charsize = tchar
endelse
endelse

if(name eq 0) then begin
class1arr = self->GetClassArray(0, 1, scaling, 1)
class2arr = self->GetClassArray(0, 1, scaling, 2)
endif else if (name eq 1) then begin
class1arr = self->GetClassArray(1, 1, scaling, 1)
```

```
class2arr = self->GetClassArray(1, 1, scaling, 2)
endif

xpos += xspacing + 8
if(name eq 0 or name eq 1) then begin
n1 = n elements(class1arr)
n2 = n elements(class2arr)
if(n elements(class1arr) gt 1 AND n elements(class2arr) gt 1) then begin
uVal = self->CalculateU(class1arr, class2arr)
boxColor = self->GetUColor(n elements(class1arr), n elements(class2arr), uVal)
maxU = n elements(class1arr)*n elements(class2arr)
if( maxU - uVal gt uVal) then uVal = maxU - uVal
xs = 46
ys = 28
xlp = xpos -19
ylp = ypos -12
x = [xlp, xlp+xs, xlp+xs, xlp]
y = [ylp, ylp, ylp+ys, ylp+ys]
if(boxColor eq foregroundColor) then boxColor = backgroundColor
POLYFILL, x, y, color = boxColor, /DEVICE
if(self->UValueSignificant(n1, n2, uVal) eq 1) then begin
xyouts, xpos, ypos, string(float(uVal), format='(F6.2)'), color = foregroundColor, alignment =
0.5, charthick = 1.6, charsize = schar, /DEVICE
endif else begin
xyouts, xpos, ypos, string(float(uVal), format='(F6.2)'), color = foregroundColor, alignment =
0.5, /DEVICE, charsize = nchar
endelse
endif
endif else begin
xyouts, xpos, ypos, 'X', color = foregroundColor, alignment = 0.5, /DEVICE, charsize = nchar
endelse
xpos += xspacing
for number = 7, 16 do begin

class1arr = self->GetClassArray(name, number, scaling, 1)
class2arr = self->GetClassArray(name, number, scaling, 2)

n1 = n elements(class1arr)
n2 = n elements(class2arr)
if(n elements(class1arr) gt 1 AND n elements(class2arr) gt 1) then begin
uVal = self->CalculateU(class1arr, class2arr)
boxColor = self->GetUColor(n elements(class1arr), n elements(class2arr), uVal)
maxU = n elements(class1arr)*n elements(class2arr)
if( maxU - uVal gt uVal) then uVal = maxU - uVal
xs = 46
ys = 28
xlp = xpos -19
ylp = ypos -12
x = [xlp, xlp+xs, xlp+xs, xlp]
y = [ylp, ylp, ylp+ys, ylp+ys]
if(boxColor eq foregroundColor) then boxColor = backgroundColor
POLYFILL, x, y, color = boxColor, /DEVICE
if(self->UValueSignificant(n1, n2, uVal) eq 1) then begin
xyouts, xpos, ypos, string(float(uVal), format='(F6.2)'), color = foregroundColor, alignment =
0.5, charthick = 1.6, charsize = schar, /DEVICE
endif else begin
xyouts, xpos, ypos, string(float(uVal), format='(F6.2)'), color = foregroundColor, alignment =
0.5, /DEVICE , charsize = nchar
endelse
xpos += xspacing
endif
endfor
END

PRO LVA TraceAnalyzer::PlotR2Line, id, xposref, yposref, xspacing
if(self.invertBackground eq 1) then begin
backgroundColor = 16777215
foregroundColor  = 0
endif else begin
backgroundColor = 0
foregroundColor  = 16777215
endelse

xpos = xposref
ypos = yposref
xyouts, xpos, ypos, ('r2 p'+string(id, format='(F4.0)')), color = foregroundColor, alignment =
```

```
0.5, /DEVICE
xpos += xspacing +40
for number = 7, 16 do begin
uVal = (*(self).traceDataArray[id])->LVA Getk(4,number)
if(number eq 14) then xyouts, xpos, ypos, string(float(sqrt(uVal)), format='(F6.3)'), color =
foregroundColor, alignment = 0.5, /DEVICE $
else xyouts, xpos, ypos, string(float(uVal), format='(F6.2)'), color = foregroundColor,
alignment = 0.5, /DEVICE
xpos += xspacing
endfor
END

PRO LVA_TraceAnalyzer::PlotScatter, id, xname, xnumber, xscaling, yname, ynumber, yscaling
if(self.invertBackground eq 1) then begin
backgroundColor = 16777215
foregroundColor  = 0
endif else begin
backgroundColor = 0
foregroundColor  = 16777215
endelse

xlabel = self->GetLabel(xname, xnumber, xscaling)
ylabel = self->GetLabel(yname, ynumber, yscaling)

if(self.scatterMode eq 1) then begin
x1arr = self->GetClassArray(xname, xnumber, xscaling, 1)
x2arr = self->GetClassArray(xname, xnumber, xscaling, 2)

y1arr = self->GetClassArray(yname, ynumber, yscaling, 1)
y2arr = self->GetClassArray(yname, ynumber, yscaling, 2)

noColor = 0
color1arr = self->GetClassColorArray(1)
for ind = 0, n elements(color1arr)-1 do if(color1arr[ind] eq backgroundColor) then
color1arr[ind] = foregroundColor
if(noColor eq 1) then for ind = 0, n_elements(color1arr)-1 do color1arr[ind] = foregroundColor

color2arr = self->GetClassColorArray(2)
for ind = 0, n elements(color2arr)-1 do if(color2arr[ind] eq backgroundColor) then
color2arr[ind] = foregroundColor
if(noColor eq 1) then for ind = 0, n_elements(color2arr)-1 do color2arr[ind] = foregroundColor

if(n elements(x1arr) gt 1) then begin
if(n elements(x2arr) gt 1) then begin
xmax = max([max(x1arr), max(x2arr)])
xmin = min([min(x1arr), min(x2arr)])
if(xmin gt 0) then xmin = 0
ymax = max([max(y1arr), max(y2arr)])
ymin = min([min(y1arr), min(y2arr)])
if(ymin gt 0) then ymin = 0
lplot, [0], [0], lOverWrite = 0, color = foregroundColor, background = backgroundColor,
psym=3, xrange = [xmin*1.05, xmax*1.05], yrange = [ymin*1.05, ymax*1.05], xtitle = xlabel,
ytitle = ylabel, charsize = 2
lplot, x1arr, y1arr, lOverWrite = 1, lcolor = color1arr, psym=4
lplot, x2arr, y2arr, lOverWrite = 1, lcolor = color2arr, psym=2
endif else begin
xmax = max(x1arr)
xmin = min(x1arr)
if(xmin gt 0) then xmin = 0
ymax = max(y1arr)
ymin = min(y1arr)
if(ymin gt 0) then ymin = 0
lplot, [0], [0], lOverWrite = 0, color = foregroundColor, background = backgroundColor,
psym=3, xrange = [xmin*1.05, xmax*1.05], yrange = [ymin*1.05, ymax*1.05], xtitle = xlabel,
ytitle = ylabel, charsize = 2
lplot, x1arr, y1arr, lOverWrite = 1, lcolor = color1arr, psym=4
endelse
endif else if(n_elements(x2arr) gt 1) then begin
xmax = max(x2arr)
xmin = min(x2arr)
if(xmin gt 0) then xmin = 0
ymax = max(y2arr)
ymin = min(y2arr)
if(ymin gt 0) then ymin = 0
lplot, [0], [0], lOverWrite = 0, color = foregroundColor, background = backgroundColor,
psym=3, xrange = [xmin*1.05, xmax*1.05], yrange = [ymin*1.05, ymax*1.05], xtitle = xlabel,
```

```
ytitle = ylabel, charsize = 2
lplot, x2arr, y2arr, lOverWrite = 1, lcolor = color2arr, psym=4
endif
;print, 'Scatter plotting ', id, xname, xnumber, xscatter, yname, ynumber, yscatter
endif
if(self.scatterMode eq 2) then begin
xarr = self->GetTumorArray(xname, xnumber, xscaling)
yarr = self->GetTumorArray(yname, ynumber, yscaling)

xmax = max(xarr)
xmin = min(xarr)
if(xmin gt 0) then xmin = 0
ymax = max(yarr)
ymin = min(yarr)
if(ymin gt 0) then ymin = 0

lplot, [0], [0], lOverWrite = 0, color = foregroundColor, background = backgroundColor,
psym=3, xrange = [xmin*1.05, xmax*1.05], yrange = [ymin*1.05, ymax*1.05], xtitle = xlabel,
ytitle = ylabel, charsize = 2
lplot, xarr, yarr, lOverWrite = 1, color = foregroundColor, psym=1, SYMSIZE=0.25
endif

END

FUNCTION LVA_TraceAnalyzer::GetScatterIndexNames
if(self.scatterMode eq 2) then begin
if(self.activeTraceData eq 0) then return, self.scaleSelectorArray
patientList = make array(self.activeTraceData, /STRING)
for pasInd = 0, self.activeTraceData -1 do patientList[pasind] = 'Pasient ' + string(pasInd+1)
return, patientList
endif
return, self.selectorArray
END

FUNCTION LVA_TraceAnalyzer::GetClassArray, name, number, scaling, class
nrClass = 0
for tDataNr = 0, self.activeTraceData -1 do begin
if(*self.traceDataArray[tDataNr]->LVA_getClass() eq class) then nrClass++
endfor
if(nrClass eq 0) then return, 0

classArr = fltarr(nrClass)

classIndex = 0

for tDataNr = 0, self.activeTraceData -1 do begin
if(*self.traceDataArray[tDataNr]->LVA_getClass() eq class) then begin
case name of
0 : begin
if(scaling eq 1) then classArr[classIndex] = *self.traceDataArray[tDataNr]->LVA_GetSUV(2,
number)/*self.traceDataArray[tDataNr]->LVA_RefTissue(2, 2)$
else classArr[classIndex] = *self.traceDataArray[tDataNr]->LVA_GetSUV(2, number)
endcase
1 : begin
if(scaling eq 1) then classArr[classIndex] = *self.traceDataArray[tDataNr]->LVA_GetSUV(45,
number)/*self.traceDataArray[tDataNr]->LVA RefTissue(45, 2)$
else classArr[classIndex] = *self.traceDataArray[tDataNr]->LVA_GetSUV(45, number)
endcase
2 : begin
if(scaling eq 1) then earlyVal = *self.traceDataArray[tDataNr]->LVA_GetSUV(2,
number)/*self.traceDataArray[tDataNr]->LVA RefTissue(2, 2)$
else earlyVal = *self.traceDataArray[tDataNr]->LVA GetSUV(2, number)
if(scaling eq 1) then lateVal = *self.traceDataArray[tDataNr]->LVA_GetSUV(45,
number)/*self.traceDataArray[tDataNr]->LVA_RefTissue(45, 2)$
else lateVal = *self.traceDataArray[tDataNr]->LVA_GetSUV(45, number)
classArr[classIndex] = max(earlyVal, lateVal)
endcase
3 : classArr[classIndex] = *self.traceDataArray[tDataNr]->LVA_Getk(1, number)
4 : classArr[classIndex] = *self.traceDataArray[tDataNr]->LVA_Getk(2, number)
5 : classArr[classIndex] = *self.traceDataArray[tDataNr]->LVA Getk(3, number)
6 : classArr[classIndex] = *self.traceDataArray[tDataNr]->LVA Getk(6, number)
7 : classArr[classIndex] = *self.traceDataArray[tDataNr]->LVA Getk(7, number)
8 : classArr[classIndex] = *self.traceDataArray[tDataNr]->LVA_Getk(8, number)
9 : classArr[classIndex] = *self.traceDataArray[tDataNr]->LVA_Getk(4, number)
10 : classArr[classIndex] = *self.traceDataArray[tDataNr]->LVA_Getk(5, number)
endcase
```

```
classIndex++
endif
endfor
return, classArr
END

FUNCTION LVA_TraceAnalyzer::GetTumorArray, name, number, scaling
lvadiv = 1
if(number eq 0 AND scaling eq 1) then lvadiv = *self.traceDataArray[name]->LVA RefTissue(2, 2)
if(number eq 1 AND scaling eq 1) then lvadiv = *self.traceDataArray[name]->LVA_RefTissue(45,
2)
return, (*self.traceDataArray[name]->LVA_GetBuffer(number))/lvadiv
END

FUNCTION LVA_TraceAnalyzer::GetClassColorArray, class
nrClass = 0
for tDataNr = 0, self.activeTraceData -1 do begin
if(*self.traceDataArray[tDataNr]->LVA_getClass() eq class) then nrClass++
endfor
if(nrClass eq 0) then return, 0

colorArr = fltarr(nrClass)

classIndex = 0

for tDataNr = 0, self.activeTraceData -1 do begin
if(*self.traceDataArray[tDataNr]->LVA getClass() eq class) then begin
colorArr[classIndex] = *self.traceDataArray[tDataNr]->LVA_getLineColor() ;GetColor()
classIndex++
endif
endfor
return, colorArr
END

FUNCTION LVA_TraceAnalyzer::GetScatterIndexNumber, mode
if(self.scatterMode eq 2) then return, self.tumorSubArray
case mode of
0 : currentArr = self.SUVSubArray
1 : currentArr = self.SUVSubArray
2 : currentArr = self.SUVSubArray
3 : currentArr = self.kSubArray
4 : currentArr = self.kSubArray
5 : currentArr = self.kSubArray
6 : currentArr = self.kSubArray
7 : currentArr = self.kSubArray
8 : currentArr = self.kSubArray
9 : currentArr = self.kSubArray
10 : currentArr = self.kSubArray
else : print, 'Unrecognized mode ',mode, ' in GetScatterIndexNumber'
endcase
return, currentArr
END

FUNCTION LVA_TraceAnalyzer::GetScatterIndexScaling, mode
if(self.scatterMode eq 2) then return, self.scaleSelectorArray
case mode of
0 : currentArr = self.scaleSelectorArray
1 : currentArr = self.scaleSelectorArray
2 : currentArr = self.scaleSelectorArray
3 : currentArr = self.emptySelectorArray
4 : currentArr = self.emptySelectorArray
5 : currentArr = self.emptySelectorArray
6 : currentArr = self.emptySelectorArray
7 : currentArr = self.emptySelectorArray
8 : currentArr = self.emptySelectorArray
9 : currentArr = self.emptySelectorArray
10 : currentArr = self.emptySelectorArray
else : print, 'Unrecognized mode ',mode, ' in GetScatterIndexScaling'
endcase
return, currentArr
END

FUNCTION LVA_TraceAnalyzer::CalculateU, class1, class2
U = 0.0
sorted1 = class1[sort(class1)]
sorted2 = class2[sort(class2)]
```

```
for ind2 = 0, n_elements(sorted2)-1 do begin
for ind1 = 0, n_elements(sorted1)-1 do begin
if(sorted2[ind2] eq sorted1[ind1]) then U += 0.5
if(sorted2[ind2] gt sorted1[ind1]) then U += 1.0
endfor
endfor
return, U
END

PRO LVA_TraceAnalyzer::ScatterPlotArray, class1arr, mode, secondClass=class2arr,
invertBackground = invB
if(KEYWORD_SET(invB) eq 0) then invB = 0
if(invB eq 1) then begin
backgroundColor = 16777215
foregroundColor  = 0
endif else begin
backgroundColor = 0
foregroundColor  = 16777215
endelse

if(class1arr[0,15] eq backgroundColor) then class1arr[0,15] = foregroundColor
if(KEYWORD_SET(class2arr)) then if(class2arr[0,15] eq backgroundColor) then class2arr[0,15] =
foregroundColor

case mode of
0   :   begin ;Max SUV early agains max SUV late
xlabel = 'SUV at 2 min'
ylabel = 'SUV at 45 min'
x1arr = class1arr[*,0]
y1arr = class1arr[*,4]
if(KEYWORD_SET(class2arr)) then begin
x2arr = class2arr[*,0]
y2arr = class2arr[*,4]
endif
endcase
1   :   begin ;Max SUV early agains max SUV late scaled to reference tissue
xlabel = 'SUV/Tissue ratio at 2 min'
ylabel = 'SUV/Tissue ratio at 45 min'
x1arr = class1arr[*,0]/class1arr[*, 13]
y1arr = class1arr[*,4]/class1arr[*, 14]
if(KEYWORD_SET(class2arr)) then begin
x2arr = class2arr[*,0]/class2arr[*, 13]
y2arr = class2arr[*,4]/class2arr[*, 14]
endif
endcase
2   :   begin ;Peak scatter
xlabel = 'SUV peak at 2 min'
ylabel = 'SUV peak at 45 min'
x1arr = class1arr[*,1]
y1arr = class1arr[*,5]
if(KEYWORD_SET(class2arr)) then begin
x2arr = class2arr[*,1]
y2arr = class2arr[*,5]
endif
endcase
3   :   begin ;Peak scatter scaled to reference tissue
xlabel = 'SUV peak/Tissue ratio at 2 min'
ylabel = 'SUV peak/Tissue ratio at 45 min'
x1arr = class1arr[*,1]/class1arr[*, 13]
y1arr = class1arr[*,5]/class1arr[*, 14]
if(KEYWORD_SET(class2arr)) then begin
x2arr = class2arr[*,1]/class2arr[*, 13]
y2arr = class2arr[*,5]/class2arr[*, 14]
endif
endcase
4   :   begin ;98Percentile scatter
xlabel = 'SUV 98 percentile at 2 min'
ylabel = 'SUV 98 percentile at 45 min'
x1arr = class1arr[*,2]
y1arr = class1arr[*,6]
if(KEYWORD_SET(class2arr)) then begin
x2arr = class2arr[*,2]
y2arr = class2arr[*,6]
endif
endcase
5   :   begin ;98Percentile scaled to reference tissue
```

```
xlabel = 'SUV 98 percentile/Tissue ratio at 2 min'
ylabel = 'SUV 98 percentile/tissue ratio at 45 min'
x1arr = class1arr[*,2]/class1arr[*, 13]
y1arr = class1arr[*,6]/class1arr[*, 14]
if(KEYWORD_SET(class2arr)) then begin
x2arr = class2arr[*,2]/class2arr[*, 13]
y2arr = class2arr[*,6]/class2arr[*, 14]
endif
endcase
6   :   begin ;K1 to k3 scatter
xlabel = '90 percentile of k1'
ylabel = '90 percentile of k2'
x1arr = class1arr[*,8]
y1arr = class1arr[*,9]
if(KEYWORD_SET(class2arr)) then begin
x2arr = class2arr[*,8]
y2arr = class2arr[*,9]
endif
endcase
7   :   begin ;Slope to intercept scatter
xlabel = 'Patlak slope'
ylabel = 'Patlak intercept'
x1arr = class1arr[*,10]
y1arr = class1arr[*,11]
if(KEYWORD_SET(class2arr)) then begin
x2arr = class2arr[*,10]
y2arr = class2arr[*,11]
endif
endcase
8   :   begin ;Metabolic to patlak scatter
xlabel = 'Metabolic value'
ylabel = 'Patlak slope'
x1arr = class1arr[*,10]
y1arr = class1arr[*,12]
if(KEYWORD_SET(class2arr)) then begin
x2arr = class2arr[*,10]
y2arr = class2arr[*,12]
endif
endcase
endcase

if(KEYWORD_SET(class2arr)) then begin
xmax = max([max(x1arr), max(x2arr)])
ymax = max([max(y1arr), max(y2arr)])
lplot, [0], [0], lOverWrite = 0, color = foregroundColor, background = backgroundColor,
psym=3, xrange = [0, xmax*1.05], yrange = [0, ymax*1.05], xtitle = xlabel, ytitle = ylabel,
charsize = 2
lplot, x1arr, y1arr, lOverWrite = 1, lcolor = class1arr[*,15], psym=4
lplot, x2arr, y2arr, lOverWrite = 1, lcolor = class2arr[*,15], psym=2
endif else begin
xmax = max(x1arr)
ymax = max(y1arr)
lplot, [0], [0], lOverWrite = 0, color = foregroundColor, background = backgroundColor,
psym=3, xrange = [0, xmax*1.05], yrange = [0, ymax*1.05], xtitle = xlabel, ytitle = ylabel,
charsize = 2
lplot, x1arr, y1arr, lOverWrite = 1, lcolor = class1arr[*,15], psym=4
endelse
END

PRO LVA_TraceAnalyzer::PlotSequence, mode

if(self.invertBackground) then begin
backgroundColor = 16777215
foregroundColor  = 0
endif else begin
backgroundColor = 0
foregroundColor  = 16777215
endelse

case mode of
1 : begin
ylabel = '98!Uth!N percentile of SUV!UE!N (2 min)'
xlabel = 'Therapy progression'
xrng = [0, 5]
yrng = [0, 10]
endcase
```

```
2 : begin
ylabel = '98!Uth!N percentile of SUV!S!UE!R!Dr!N (2 min)'
xlabel = 'Therapy progression'
xrng = [0, 5]
yrng = [0, 11]
endcase
3 : begin
ylabel = '98!Uth!N percentile of SUV!UL!N (45 min)'
xlabel = 'Therapy progression'
xrng = [0, 5]
yrng = [0, 15]
endcase
4 : begin
ylabel = '98!Uth!N percentile of SUV!S!UL!R!Dr!N (45 min)'
xlabel = 'Therapy progression'
xrng = [0, 5]
yrng = [0, 17]
endcase
5 : begin
ylabel = '98!Uth!N percentile of K1'
xlabel = 'Therapy progression'
xrng = [0, 5]
yrng = [0, 1.2]
endcase
6 : begin
ylabel = '98!Uth!N percentile of the MR!DFDG'
xlabel = 'Therapy progression'
xrng = [0, 5]
yrng = [0, 0.15]
endcase
7 : begin
ylabel = '98!Uth!N percentile of the vascular fraction'
xlabel = 'Therapy progression'
xrng = [0, 5]
yrng = [0, 0.3]
endcase
8 : begin
ylabel = '!7r!3/Peak'
xlabel = 'Therapy progression'
xrng = [0, 5]
yrng = [0.1, 0.25]
endcase
endcase

lplot, [0], [0], lOverWrite = 0, color = foregroundColor, background = backgroundColor,
psym=3, xrange = xrng, yrange = yrng, xtitle = xlabel, ytitle = ylabel, charsize = 2

for sindex = 0, 3 do begin
nrVals = 0
for ind = 0, 3 do if(ptr valid(self.sequencesArray[sindex,ind]) eq 1) then nrVals++
xarr = fltarr(nrVals)
yarr = fltarr(nrVals)

current = 0
for ind = 0, 3 do begin
if(ptr valid(self.sequencesArray[sindex,ind]) eq 1) then begin
case mode of
1 : yarr[current] = (*self.sequencesArray[sindex,ind])->LVA_GetSUV(2,8)
2 : yarr[current] = (*self.sequencesArray[sindex,ind])-
>LVA GetSUV(2,8)/(*self.sequencesArray[sindex,ind])->LVA RefTissue(2,2)
3 : yarr[current] = (*self.sequencesArray[sindex,ind])->LVA_GetSUV(45,8)
4 : yarr[current] = (*self.sequencesArray[sindex,ind])-
>LVA_GetSUV(45,8)/(*self.sequencesArray[sindex,ind])->LVA_RefTissue(45,2)
5 : yarr[current] = (*self.sequencesArray[sindex,ind])->LVA_Getk(1, 8)
6 : yarr[current] = (*self.sequencesArray[sindex,ind])->LVA Getk(6, 8)
7 : yarr[current] = (*self.sequencesArray[sindex,ind])->LVA Getk(5, 8)
8 : yarr[current] = (*self.sequencesArray[sindex,ind])->LVA_GetEMHetrogenity()
endcase
xarr[current] = ind+1
current++
endif
endfor
mysym = -(sindex+1)
if(mysym eq -3) then mysym = -7
lplot, xarr, yarr, lOverWrite = 1, color = foregroundColor, psym = mysym
endfor
```

```
carr = [foregroundColor, foregroundColor, foregroundColor, foregroundColor]
legend,['Pasient 2','Pasient 5','Pasient 9', 'Pasient10'],psym=[2,1,7,4], colors = carr,
textcolors = carr, box = 0, charsize = 1.5
END

;;;;;;;;;;;;;;;;
;; Destructor ;;
;;;;;;;;;;;;;;;;

PRO LVA_TraceAnalyzer::Cleanup
self->LVA_EraseBuffers
END

PRO LVA_TraceAnalyzer::LVA_EraseBuffers
for index = 0, (size(traceDataArray))[0] -1 do begin
OBJ DESTROY, *self.traceDataArray[index]
if(ptr_valid(self.traceDataArray[index]) eq 1) then ptr_free, self.traceDataArray[index]
endfor
END
```

Trace data object

```
PRO LVA_TraceData__define
struct =$
{$
LVA_TraceData, $
name:'Default', $
k1BinSize:0.12,$
k2BinSize:0.12,$
k3BinSize:0.012,$
pearsonBinSize:0.01,$
k1Ptr:ptr_new(),$
k2Ptr:ptr_new(),$
k3Ptr:ptr new(),$
vbPtr:ptr new(),$
mbPtr:ptr new(),$
pearsonPtr:ptr new(),$
pSlPtr:ptr_new(),$
pInPtr:ptr_new(),$
plasmaPtr:ptr new(),$
timePtr:ptr new(),$
tumorPtr:ptr_new(),$
tumorPeakPtr:ptr_new(),$
tumorBinSize:0.1,$
tissuePtr:ptr new(),$
tissueBinSize:0.1,$
patlakSlopeBinSize:0.0,$
patlakSlopeShift:0.0,$
patlakInterceptBinSize:0.0,$
patlakInterceptShift:0.0,$
metabolicBinSize:0.0,$
metabolicPtr:ptr_new(),$
timeindex:0L,$
lineColor:0L,$
class1Color:0L,$
class2Color:0L,$
singleBin:0L,$
classType:1L,$
classMode:1L,$
diferentialCode:0L,$
locationCode:0L,$
yearOfBirth:0L,$
sliceSpacing:0L,$
invertBackground:0L$
}
END

PRO LVA_TraceData::LVA_Constructor
self.k1BinSize=0.15
self.k2BinSize=0.21
self.k3BinSize=0.012
self.pearsonBinSize = 0.1
self.tumorBinSize = 0.5
```

```
self.tissueBinSize = 0.1
self.singleBin = 1
self.classType = 1
self.invertBackground = 0
self.class1Color = lhex2dec('FFFFFF'); White
self.class2Color = lhex2dec('0000FF'); Red
END

FUNCTION LVA_TraceData::LVA_GetBuffer, type
lastindex = (size(*self.tumorPtr, /dimensions))[1]-1
case type of
0 : return, (*self.tumorPtr)[*,7]
1 : return, (*self.tumorPtr)[*,lastindex]
2 : return, *self.k1Ptr
3 : return, *self.k2Ptr
4 : return, *self.k3Ptr
5 : return, *self.mbPtr
6 : return, *self.pSlPtr
7 : return, *self.pInPtr
8 : return, *self.vbPtr
endcase
END

FUNCTION LVA_TraceData::LVA_GetCorrelation, buf1, buf2
if(buf1 eq buf2) then return, 'X'
lastindex = (size(*self.tumorPtr, /dimensions))[1]-1

case buf1 of
0 : firstbuffer =  average((*self.tumorPtr)[*,6:8], 2)
1 : firstbuffer =  (*self.tumorPtr)[*,lastindex]
2 : firstbuffer =  *self.k1Ptr
3 : firstbuffer =  *self.k2Ptr
4 : firstbuffer =  *self.k3Ptr
5 : firstbuffer =  *self.vbPtr
6 : firstbuffer =  *self.mbPtr
7 : firstbuffer =  *self.pSlPtr
8 : firstbuffer =  *self.pInPtr
endcase
case buf2 of
0 : secondbuffer =  average((*self.tumorPtr)[*,6:8], 2)
1 : secondbuffer =  (*self.tumorPtr)[*,lastindex]
2 : secondbuffer =  *self.k1Ptr
3 : secondbuffer =  *self.k2Ptr
4 : secondbuffer =  *self.k3Ptr
5 : secondbuffer =  *self.vbPtr
6 : secondbuffer =  *self.mbPtr
7 : secondbuffer =  *self.pSlPtr
8 : secondbuffer =  *self.pInPtr
endcase
return, (correlate(firstbuffer, secondbuffer))^2
END

PRO LVA_TraceData::LVA_SetClassMode, mode
self.classMode = mode
case mode of
0:
1: self.classType = self.diferentialCode
2: if(self.yearOfBirth gt 60) then self.classType = 2 else self.classType = 1
3: if((size(*self.tumorPtr, /dimensions))[0]*self.sliceSpacing lt 1600*4) then self.classType
= 2 else self.classType = 1
4: self.classType = self.locationCode

endcase
END

PRO LVA_TraceData::LVA_Break
stop
END

PRO LVA_Tracedata::LVA_setLineColor, value
self.lineColor = value
END

FUNCTION LVA_Tracedata::LVA_getLineColor
return, self.lineColor
END
```

```
PRO LVA_TraceData::LVA_SetTime, value
if(ptr_valid(self.timePtr) eq 1) then begin
self.timeindex = float(value)/50.0*(( size(*self.timePtr, /dimensions))[0] -1)
endif
END

PRO LVA_TraceData::LVA_ToggleInvertBackground
if(self.invertBackground eq 0) then self.invertBackground = 1 else self.invertBackground = 0
END

FUNCTION LVA_TraceData::LVA_getName
if(self.name eq '') then return, ''
;Split the filepath elements
pathElements = strsplit(self.name, '\', /Extract)
;Select last filepath element and split the rider (.trc)
fileElements = strsplit(pathElements[(size(pathElements, /dimensions))[0]-1], '.', /Extract)
return, fileElements[0]
END

FUNCTION LVA_TraceData::LVA_getClass
return, self.classType
END

FUNCTION LVA_TraceData::LVA_getKpercentile, nrk, percentile, excludeZero = exz
if(KEYWORD_SET(exz) eq 0) then exz = 0
value = 0.0
case nrk of
1: kPtr = self.k1Ptr
2: kPtr = self.k2Ptr
3: kPtr = self.k3Ptr
4: kptr = self.pearsonPtr
5: kptr = self.vbPtr
6: kptr = self.mbPtr
7: kptr = self.pSlPtr
8: kptr = self.pInPtr
endcase
if(ptr_valid(kPtr) eq 1) then begin
if(exz eq 1) then begin
nonzero = (*kPtr)[where(*kPtr gt 0)]
sortedk = (nonzero)[sort(nonzero)]
value = sortedk[(n_elements(sortedk)-1)*percentile]
endif else begin
sortedk = (*kPtr)[sort(*kPtr)]
value = sortedk[(n_elements(sortedk)-1)*percentile]
endelse
endif
return, value
END

FUNCTION LVA_TraceData::LVA_GetEMHetrogenity
lpeak = self->LVA_GetSUV(45, 1)
lvar = self->LVA_GetSUV(45, 4)
lsig = sqrt(lvar)
return, lsig/lpeak
END

FUNCTION LVA_TraceData::LVA_Getk, nrk, mode
result = 0
case nrk of
1: kPtr = self.k1Ptr
2: kPtr = self.k2Ptr
3: kPtr = self.k3Ptr
4: kptr = self.pearsonPtr
5: kptr = self.vbPtr
6: kptr = self.mbPtr
7: kptr = self.pSlPtr
8: kptr = self.pInPtr
endcase

case mode of
0 : result =self->LVA getKpercentile(nrk, 0.5, excludeZero = 1)
1 : result =self->LVA_getKpercentile(nrk, 0.9, excludeZero = 1)
2 : result = (MOMENT((*kPtr)[where(*kPtr gt 0)]))[1]
3 : result = (MOMENT((*kPtr)[where(*kPtr gt 0)]))[2]
4 : result = (MOMENT((*kPtr)[where(*kPtr gt 0)]))[3]
```

```
5 : begin
lastindex = (size(*self.tumorPtr, /dimensions))[1]-1
sorted =(*self.tumorPtr)[sort((*self.tumorPtr)[*,lastindex]), lastindex]
limit = sorted[lastindex*0.95]
result = (MOMENT((*kPtr)[where((*self.tumorPtr)[*,lastindex] gt limit)]))[0]
endcase

;Matrix order
7:  result = max(*kPtr)
8:  result = self->LVA getKpercentile(nrk, 0.98, excludeZero = 1)
9:  result = self->LVA getKpercentile(nrk, 0.95, excludeZero = 1)
10: result = self->LVA_getKpercentile(nrk, 0.90, excludeZero = 1)
11: result = self->LVA_getKpercentile(nrk, 0.7, excludeZero = 1)
12: result = self->LVA_getKpercentile(nrk, 0.5, excludeZero = 1)
13: result = self->LVA getKpercentile(nrk, 0.3, excludeZero = 1)
14: result = (MOMENT((*kPtr)[where(*kPtr gt 0)]))[1]
15: result = (MOMENT((*kPtr)[where(*kPtr gt 0)]))[2]
16: result = (MOMENT((*kPtr)[where(*kPtr gt 0)]))[3]
endcase
return, result
END

FUNCTION LVA_TraceData::LVA_GetSUV, time, mode
lastindex = (size(*self.tumorPtr, /dimensions))[1]-1
interval = lindgen(lastindex+1)
if(time eq 1) then interval = lindgen(5) + 3
if(time eq 2) then interval = lindgen(5) + 6
if(time eq 45) then interval = lindgen(5) + lastindex-4
SUVValue = 0
lengthFiller = replicate(1, (size(*self.tumorPtr, /dimensions))[0])
decayAdjustment = exp((*self.timePtr)[interval]*0.0063128)##lengthFiller
case mode of
0: SUVValue = max((*self.tumorPtr)[*, interval]*decayAdjustment)
1: SUVValue = max((*self.tumorPeakPtr)[*, interval]*decayAdjustment)
2: begin
maxPercentile = 0
for index = 0, n_elements(interval) -1 do begin
sortedValues = (*self.tumorPtr)[sort((*self.tumorPtr)[*,interval[index]]), interval[index]]
indexPercentile = ((size(*self.tumorPtr, /dimensions))[0]-1.0)*0.98
currentPercentile =
sortedValues[indexPercentile]*exp((*self.timePtr)[interval[index]]*0.0063128)
maxPercentile += currentPercentile
endfor
SUVValue = maxPercentile/n_elements(interval)
endcase
3: begin
tumorMoments = MOMENT((*self.tumorPtr)[*, interval]*decayAdjustment)
SUVValue = tumorMoments[0]
endcase
4: begin
tumorMoments = MOMENT((*self.tumorPtr)[*, interval]*decayAdjustment)
SUVValue = tumorMoments[1]
endcase
5: begin
tumorMoments = MOMENT((*self.tumorPtr)[*, interval]*decayAdjustment)
SUVValue = tumorMoments[2]
endcase
6: begin
tumorMoments = MOMENT((*self.tumorPtr)[*, interval]*decayAdjustment)
SUVValue = tumorMoments[3]
endcase
;Matrix order
7: SUVValue = max((*self.tumorPtr)[*, interval]*decayAdjustment)
8: begin
maxPercentile = 0
for index = 0, n_elements(interval) -1 do begin
sortedValues = (*self.tumorPtr)[sort((*self.tumorPtr)[*,interval[index]]), interval[index]]
indexPercentile = ((size(*self.tumorPtr, /dimensions))[0]-1.0)*0.98
currentPercentile =
sortedValues[indexPercentile]*exp((*self.timePtr)[interval[index]]*0.0063128)
maxPercentile += currentPercentile
endfor
SUVValue = maxPercentile/n_elements(interval)
endcase
9: begin
maxPercentile = 0
```

```
for index = 0, n_elements(interval) -1 do begin
sortedValues = (*self.tumorPtr)[sort((*self.tumorPtr)[*,interval[index]]), interval[index]]
indexPercentile = ((size(*self.tumorPtr, /dimensions))[0]-1.0)*0.95
currentPercentile =
sortedValues[indexPercentile]*exp((*self.timePtr)[interval[index]]*0.0063128)
maxPercentile += currentPercentile
endfor
SUVValue = maxPercentile/n_elements(interval)
endcase
10: begin
maxPercentile = 0
for index = 0, n_elements(interval) -1 do begin
sortedValues = (*self.tumorPtr)[sort((*self.tumorPtr)[*,interval[index]]), interval[index]]
indexPercentile = ((size(*self.tumorPtr, /dimensions))[0]-1.0)*0.90
currentPercentile =
sortedValues[indexPercentile]*exp((*self.timePtr)[interval[index]]*0.0063128)
maxPercentile += currentPercentile
endfor
SUVValue = maxPercentile/n_elements(interval)
endcase
11: begin
maxPercentile = 0
for index = 0, n_elements(interval) -1 do begin
sortedValues = (*self.tumorPtr)[sort((*self.tumorPtr)[*,interval[index]]), interval[index]]
indexPercentile = ((size(*self.tumorPtr, /dimensions))[0]-1.0)*0.70
currentPercentile =
sortedValues[indexPercentile]*exp((*self.timePtr)[interval[index]]*0.0063128)
maxPercentile += currentPercentile
endfor
SUVValue = maxPercentile/n_elements(interval)
endcase
12: begin
maxPercentile = 0
for index = 0, n_elements(interval) -1 do begin
sortedValues = (*self.tumorPtr)[sort((*self.tumorPtr)[*,interval[index]]), interval[index]]
indexPercentile = ((size(*self.tumorPtr, /dimensions))[0]-1.0)*0.5
currentPercentile =
sortedValues[indexPercentile]*exp((*self.timePtr)[interval[index]]*0.0063128)
maxPercentile += currentPercentile
endfor
SUVValue = maxPercentile/n_elements(interval)
endcase
13: begin
maxPercentile = 0
for index = 0, n_elements(interval) -1 do begin
sortedValues = (*self.tumorPtr)[sort((*self.tumorPtr)[*,interval[index]]), interval[index]]
indexPercentile = ((size(*self.tumorPtr, /dimensions))[0]-1.0)*0.30
currentPercentile =
sortedValues[indexPercentile]*exp((*self.timePtr)[interval[index]]*0.0063128)
maxPercentile += currentPercentile
endfor
SUVValue = maxPercentile/n_elements(interval)
endcase
14: begin
tumorMoments = MOMENT((*self.tumorPtr)[*, interval]*decayAdjustment)
SUVValue = tumorMoments[1]
endcase
15: begin
tumorMoments = MOMENT((*self.tumorPtr)[*, interval]*decayAdjustment)
SUVValue = tumorMoments[2]
endcase
16: begin
tumorMoments = MOMENT((*self.tumorPtr)[*, interval]*decayAdjustment)
SUVValue = tumorMoments[3]
endcase
endcase
return, SUVValue
END

FUNCTION LVA_TraceData::LVA_RefPercentile, percentile
nrTimeIndexes = (size(*self.tissuePtr, /dimensions))[1]
RefArray = fltarr(nrTimeIndexes)
for timeIndex = 0, nrTimeIndexes -1 do begin
RefArray[timeIndex] = self->LVA_getReferenceTissueLevel(timeIndex,percentile, 2)
endfor
return, RefArray
```

```
END

FUNCTION LVA_TraceData::LVA_RefTissue, time, mode
lastindex = (size(*self.tissuePtr, /dimensions))[1]-1
interval = lindgen(lastindex+1)
if(time eq 2) then interval = lindgen(5) + 6
if(time eq 45) then interval = lindgen(5) + lastindex-4
SUVValue = 0
;lengthFiller = replicate(1, (size(*self.tissuePtr, /dimensions))[0])
;decayAdjustment = exp((*self.timePtr)[interval]*0.0063128)##lengthFiller
case mode of
0: begin
for intervalindex = 0, 4 do begin
maxSUV = max((*self.tissuePtr)[*,
interval[intervalindex]])*exp((*self.timePtr)[interval[intervalindex]]*0.0063128)
if(SUVValue lt maxSUV) then SUVValue = maxSUV
endfor
endcase
2: begin
Percentile = 0
for intervalindex = 0, 4 do begin
currentTissue = (*self.tissuePtr)[*, interval[intervalindex]]
nrTissue = n_elements(currentTissue)
sortedTissue = currentTissue[sort(currentTissue)]
Percentile += sortedTissue[(nrTissue-
1)*0.5]*exp((*self.timePtr)[interval[intervalindex]]*0.0063128)
endfor
SUVValue = Percentile/5.
endcase
endcase
return, SUVValue
END

FUNCTION LVA_TraceData::LVA_getReferenceTissueLevel, time, percentile, smoothValue
if(time gt (size(*self.tumorPtr, /dimensions))[1]-1-smoothValue) then begin
mytime =  (size(*self.tumorPtr, /dimensions))[1]-1-2*smoothValue
endif else mytime = time-smoothValue
if(mytime lt 0) then mytime = 0
summedPercentile = 0.0

for tindex = mytime, mytime+2*smoothValue do begin
currentTissue = (*self.tissuePtr)[*,tindex]
nrTissue = n elements(currentTissue)
sortedTissue = currentTissue[sort(currentTissue)]
summedPercentile += sortedTissue[(nrTissue-
1)*percentile]*exp((*self.timePtr)[tindex]*0.0063128)
endfor
return, summedPercentile/float(1.0+2.0*smoothValue)
END

FUNCTION LVA_TraceData::LVA_ReadKHistogram, traceFile
ReadU, traceFile, nrBins
if(round(nrBins) gt 0) then begin
KvaluePtr = ptr_new(fltarr(nrBins))
for index = 0, nrBins-1 do begin
ReadU, traceFile, kvalue
(*KvaluePtr)[index] = kvalue
endfor
endif else begin
KvaluePtr = ptr_new()
print, 'Missing k histogram'
endelse
return, KvaluePtr
END

FUNCTION LVA_TraceData::LVA_ReadValuegram, tracefile
ReadU, traceFile, nrkValues
if(nrkValues gt 0) then begin
valuePtr = ptr_new(fltarr(nrkValues))
for index = 0, nrkValues-1 do begin
ReadU, traceFile, kvalue
(*valuePtr)[index] = kvalue
endfor
endif else begin
valuePtr = ptr_new()
endelse
```

```
return, valuePtr
END

PRO LVA_TraceData::LVA_LoadTrace, savefile, timeIndex
if(savefile eq '') then return
OPENR, traceFile, savefile, /GET_LUN, ERROR = err

if(err eq 0) then begin
if(EOF(traceFile) ne 1) then begin
ReadU, traceFile, version

if(version gt 10054.032) then begin
self->LVA_EraseBuffers
self.name = savefile

self.k1Ptr = self->LVA ReadValuegram(traceFile)
self.k2Ptr = self->LVA_ReadValuegram(traceFile)
self.k3Ptr = self->LVA_ReadValuegram(traceFile)

if(version gt 10054.034) then self.vbPtr = self->LVA_ReadValuegram(traceFile)
if(version gt 10054.033) then self.pearsonPtr = self->LVA_ReadValuegram(traceFile)
if(version gt 10054.035) then self.pSlPtr = self->LVA_ReadValuegram(traceFile)
if(version gt 10054.035) then self.pInPtr = self->LVA_ReadValuegram(traceFile)

ReadU, traceFile, imageIntensityScale
ReadU, traceFile, weight
ReadU, traceFile, injectedActivity

if(version gt 10054.035) then begin
        ReadU, traceFile, tumorLocationCode
        ReadU, traceFile, tumorTypeCode
        ReadU, traceFile, yearOfBirth
        ReadU, traceFile, sliceSpacing
        self.diferentialCode = tumorTypeCode
        self.locationCode = tumorLocationCode
        self.yearOfBirth = yearOfBirth
        self.sliceSpacing = sliceSpacing
endif

ReadU, traceFile, nrPlasma
self.plasmaPtr = ptr_new(fltarr(nrPlasma))
for pindex = 0, nrPlasma -1 do begin
        if(EOF(traceFile) ne 1) then ReadU, traceFile, plasmaValue
        (*self.plasmaPtr)[pindex] =
plasmaValue*imageIntensityScale*weight*1000/injectedActivity
endfor

ReadU, traceFile, nrTimevalues
self.timePtr = ptr new(fltarr(nrTimevalues))
for tindex = 0, nrTimevalues -1 do begin
        if(EOF(traceFile) ne 1) then ReadU, traceFile, timeValue
        (*self.timePtr)[tindex] = timeValue
endfor

; Tumor histogram
ReadU, traceFile, nrTumorValues
ReadU, traceFile, nrTumorTimeValues
self.tumorPtr = ptr_new(fltarr(nrTumorValues, nrTimeValues))
for tumindex = 0, nrTumorValues-1 do begin
        for tindex = 0, nrTumorTimeValues -1 do begin
                if(EOF(traceFile) ne 1) then ReadU, traceFile, tumorValue
                (*self.tumorPtr)[tumindex, tindex] = tumorValue
        endfor
endfor

self.tumorPeakPtr = ptr new(fltarr(nrTumorValues, nrTimeValues))
for tumindex = 0, nrTumorValues-1 do begin
        for tindex = 0, nrTumorTimeValues -1 do begin
                if(EOF(traceFile) ne 1) then ReadU, traceFile, tumorValue
                (*self.tumorPeakPtr)[tumindex, tindex] = tumorValue
        endfor
endfor

;Tissue values
if(version gt 10054.035) then begin
        ReadU, traceFile, nrTissuePoints
```

```
        self.tissuePtr = ptr_new(fltarr(nrTissuePoints, nrTimevalues))
        for tisindex = 0, nrTissuePoints-1 do begin
                for tindex = 0, nrTimevalues -1 do begin
                        if(EOF(traceFile) ne 1) then ReadU, traceFile, tissueValue
                        (*self.tissuePtr)[tisindex, tindex] = tissueValue
                endfor
        endfor
 endif else begin
        ReadU, traceFile, nrTissueBins
        ReadU, traceFile, binSize
        self.tissueBinSize = binSize
        self.tissuePtr = ptr_new(fltarr(nrTissueBins, nrTimevalues))
        for tisindex = 0, nrTissuebins-1 do begin
                for tindex = 0, nrTimevalues -1 do begin
                        if(EOF(traceFile) ne 1) then ReadU, traceFile, tissueValue
                        (*self.tissuePtr)[tisindex, tindex] = tissueValue
                endfor
        endfor
endelse

;The patlak slope and intercept
ReadU, traceFile, dim1
ReadU, traceFile, dim2
;self.patlakVals = ptr_new(fltarr(dim1, dim2))
for ind1 = 0L, dim1 -1 do begin
        for ind2 = 0L, dim2 -1 do begin
                ReadU, traceFile, patlakValue
                ;(*self.patlakVals)[ind1, ind2] = patlakValue
        endfor
endfor

self.patlakSlopeBinSize = 0.05
self.patlakInterceptBinSize = 1.0
;
;The matabolic rate values
self.mbPtr = self->LVA_ReadValuegram(traceFile)

nrBins = 100.0
self.metabolicBinSize = 0.002
nrBins = max(*self.mbPtr)/self.metabolicBinSize +1

self.metabolicPtr = ptr_new(fltarr(nrBins))

binValues = floor((*self.mbPtr)/self.metabolicBinSize)

for mpoint = 0L, n_elements(binValues) -1 do begin
        (*self.metabolicPtr)[binValues[mpoint]]++
endfor

ReadU, traceFile, tail

self->LVA_SetTime, timeIndex
self->LVA_SetClassMode, 0
if(tail ne 3703.0) then print, 'Error in reading file'
endif else begin
print, 'Unsuported savefile'
endelse
endif else begin
print, 'Empty savefile'
endelse
endif else begin
print, 'Could not open file'
endelse

CLOSE, traceFile
FREE_LUN, traceFile
END

PRO LVA_TraceData::PrintLegendHeader, xpos, ypos
if(self.invertBackground eq 1) then begin
backgroundColor = 16777215
foregroundColor  = 0
endif else begin
backgroundColor = 0
foregroundColor  = 16777215
endelse
```

```
;clearBuffer = fltarr(600,600)
clearBuffer = replicate(backgroundColor, 600, 600)

tv, clearBuffer
xyouts, xpos, ypos, 'Name', /DEVICE
xpos += 85
xyouts, xpos, ypos, 'Ear', color = foregroundColor, alignment = 0.5, /DEVICE
xpos += 37
xyouts, xpos, ypos, 'Eref', color = foregroundColor, alignment = 0.5, /DEVICE
xpos += 38
xyouts, xpos, ypos, 'Late', color = foregroundColor, alignment = 0.5, /DEVICE
xpos += 38
xyouts, xpos, ypos, 'Lref', color = foregroundColor, alignment = 0.5, /DEVICE
xpos += 41
xyouts, xpos, ypos, 'k1', color = foregroundColor, alignment = 0.5, /DEVICE
xpos += 37
xyouts, xpos, ypos, 'k2', color = foregroundColor, alignment = 0.5, /DEVICE
xpos += 37
xyouts, xpos, ypos, 'k3', color = foregroundColor, alignment = 0.5, /DEVICE
xpos += 39
xyouts, xpos, ypos, 'met', color = foregroundColor, alignment = 0.5, /DEVICE
xpos += 39
xyouts, xpos, ypos, 'pat', color = foregroundColor, alignment = 0.5, /DEVICE
xpos += 35
xyouts, xpos, ypos, 'vb', color = foregroundColor, alignment = 0.5, /DEVICE
END

FUNCTION LVA_TraceData::GetNameStr
pathElements = strsplit(self.name, '\', /Extract)
fileElements = strsplit(pathElements[(size(pathElements, /dimensions))[0]-1], '.', /Extract)
return, fileElements[0]
END

FUNCTION LVA_Tracedata::GetColor
if(self.classType eq 1) then return, self.class1Color
if(self.classType eq 2) then return, self.class2Color
return, 0
END

PRO LVA_TraceData::PrintLegend, xpos, ypos, silent

namstr = self->GetNameStr()

lastindex = (size(*self.tumorPtr, /dimensions))[1]-1
lateSuv = (*self.tumorPtr)[*, lastindex]*exp((*self.timePtr)[lastindex]*0.0063128)

print, 'mb to pat' , string((correlate(*self.mbPtr, *self.pSlPtr))^2)
print, 'SUV to pat' , string((correlate(lateSuv, *self.pSlPtr))^2)
print, 'mb to SUV' , string((correlate(*self.mbPtr, lateSuv))^2)

;; new
early98 = self->LVA_GetSUV(2, 8)
early98Ref = self->LVA_GetSUV(2, 8)/self->LVA_RefTissue(2, 2)
lateMax = self->LVA GetSUV(45, 8)
lateMaxRef = self->LVA GetSUV(45, 8)/self->LVA_RefTissue(45, 2)
k1Mask = self->LVA_Getk(1, 8)
k2Mask = self->LVA_Getk(2, 8)
k3Mask = self->LVA Getk(3, 8)
metabo = self->LVA Getk(6, 8)
pslope = self->LVA Getk(7, 8)
vbMask = self->LVA_Getk(5, 8)
;; Old

tindex = (size(*self.tumorPtr, /dimensions))[1]-1
sortedValues = (*self.tumorPtr)[sort((*self.tumorPtr)[*,tindex]), tindex]
indexPercentile = ((size(*self.tumorPtr, /dimensions))[0]-1.0)*0.98
tumorPercent = sortedValues[indexPercentile]*exp((*self.timePtr)[tindex]*0.0063128)

tumorMoments = MOMENT((*self.tumorPtr)[*, (size(*self.tumorPtr, /dimensions))[1]-1])
meanSUV = tumorMoments[0]*exp((*self.timePtr)[tindex]*0.0063128)

if(strlen(namstr) eq 8) then outstr = namstr+' ' else outstr = namstr
if(silent eq 0) then begin
tumostr = string(max(*self.tumorPtr)*exp((*self.timePtr)[tindex]*0.0063128), format='(F6.2)')
peastr = string(max(*self.tumorPeakPtr)*exp((*self.timePtr)[tindex]*0.0063128),
```

```
format='(F6.2)')
perstr = string(tumorPercent, format='(F6.2)')
meastr = string(meanSUV, format='(F6.2)')

;tisstr = string(tissueMedian, format='(F6.2)')
metstr = string(metabo, format='(F6.3)')
slostr = string(pslope, format='(F6.2)')

pk1str = string((*self.k1Ptr)[(sort(*self.k1Ptr))[0.95*(n_elements(*self.k1Ptr)-1)]],
format='(F6.2)')
pk3str = string((*self.k3Ptr)[(sort(*self.k3Ptr))[0.95*(n_elements(*self.k3Ptr)-1)]],
format='(F6.3)')

sizstr = string((size(*self.tumorPtr, /dimensions))[0], format='(F8.1)')

early98str =  string(early98, format='(F6.2)')
early98Refstr =  string(early98Ref, format='(F6.2)')
lateMaxstr =  string(lateMax, format='(F6.2)')
lateMaxRefstr =  string(lateMaxRef, format='(F6.2)')
k1Maskstr =  string(k1Mask, format='(F6.2)')
k2Maskstr =  string(k2Mask, format='(F6.2)')
k3Maskstr =  string(k3Mask, format='(F6.3)')
metabostr =  string(metabo, format='(F6.3)')
pslopestr =  string(pslope, format='(F6.2)')
vbMaskstr =  string(vbMask, format='(F6.2)')

outstr +=
early98str+early98Refstr+lateMaxstr+lateMaxRefstr+k1Maskstr+k2Maskstr+k3Maskstr+metabostr+pslo
pestr+vbMaskstr
;outstr += tumostr + peastr + perstr + meastr + tisstr + metstr + slostr + pk1str + pk3str +
sizstr
endif

if(self.invertBackground eq 1 AND self.lineColor eq 16777215) then begin
xyouts, xpos, ypos, outstr, color = 0, /DEVICE
endif else begin if(self.invertBackground eq 0 AND self.lineColor eq 0) then begin
xyouts, xpos, ypos, outstr, color = 16777215, /DEVICE
endif else xyouts, xpos, ypos, outstr, color = self.lineColor, /DEVICE
endelse

END

PRO LVA_TraceData::PlotK1, maxx, maxy, overWrite
self->plotHistogram, *self.k1Ptr, self.k1BinSize, maxx, maxy, 'k1', 'Normalized count',
overWrite
END

PRO LVA_TraceData::PlotK2, maxx, maxy, overWrite
self->plotHistogram, *self.k2Ptr, self.k2BinSize, maxx, maxy, 'k2', 'Normalized count',
overWrite
END

PRO LVA_TraceData::PlotK3, maxx, maxy, overWrite
self->plotHistogram, *self.k3Ptr, self.k3BinSize, maxx, maxy, 'k3', 'Normalized count',
overWrite
END

PRO LVA_TraceData::PlotR2, maxx, maxy, overWrite
self->plotHistogram, *self.pearsonPtr, self.pearsonBinSize, maxx, maxy, 'r2', 'Normalized
count', overWrite
END

PRO LVA_TraceData::PlotPatlak, type, maxx, maxy, overWrite
piBinSize = 4
psBinSize = 0.1

if(type eq 'i') then self->plotHistogram, *self.pInPtr, piBinSize, maxx, maxy, 'Patlak
intercept', 'Normalized count', overWrite
if(type eq 's') then self->plotHistogram, *self.pSlPtr, psBinSize, maxx, maxy, 'Patlak slope',
'Normalized count', overWrite
END

PRO LVA_TraceData::PlotPlasma, maxx, maxy, overWrite
if(overWrite eq 0) then begin
plot, *self.timePtr, *self.plasmaPtr, xrange = [0, maxx], yrange = [0, maxy]
endif else oplot, *self.timePtr, *self.plasmaPtr, color = self.lineColor
```

```
END

PRO LVA_TraceData::plotMBHistogram
self->LVA_PlotHistogram, self.mbPtr, 'Metabolic rate', 'Normalized count'
END

PRO LVA_TraceData::plotVBHistogram
self->LVA_PlotHistogram, self.vbPtr, 'Vascular fraction', 'Normalized count'
END

PRO LVA_TraceData::plotPatlakSlopeHistogram
self->LVA_PlotHistogram, self.pSlPtr, 'Patlak slope', 'Normalized count'
END

PRO LVA_TraceData::plotPatlakInterceptHistogram
self->LVA_PlotHistogram, self.pInPtr, 'Patlak intercept','Normalized count'
END

PRO LVA_TraceData::LVA_PlotHistogram, dataPtr, xlabel, ylabel
if(self.invertBackground eq 1) then begin
backgroundColor = 16777215
foregroundColor  = 0
endif else begin
backgroundColor = 0
foregroundColor  = 16777215
endelse
binsize = (max(*dataPtr) - min(*dataPtr)) * 0.05
;Our histogram
if(ptr_valid(dataPtr) eq 1) then begin
sub = (*dataPtr)
if(max(sub) gt 0) then begin
hindex = where(sub gt 0) ;max(sub))
if(n_elements(hindex) gt 0) then begin
if((size(hindex))[1] gt 0) then begin
       histRange = imclip(sub[hindex], PERCENT= 100)
       hist_plot, sub[hindex], /normal, /fill, xstyle=3, background = backgroundColor, color =
foregroundColor, xtitle = xlabel, ytitle = ylabel, charsize = 2, binsize = binsize, xticks = 3
       ;oplot, [histRange[1], histRange[1]], [0.0, 1.0], linestyle=2
endif
endif
endif
endif
END

PRO LVA_TraceData::plotHistogram, data, binsize, maxx, maxy, xlabel, ylabel, overWrite
;hist_plot, data, /normal, /fill, xstyle=3, color = 0, background = 16777215, max = maxx/4.,
binsize = maxx*0.01, xtitle = xlabel, ytitle = ylabel, charsize = 2
;return
binValues = floor(data/binsize)
if(min(binValues) lt 0) then histShift = min(binValues) else histShift = 0
print, histShift
binValues -= histShift
nrBins = max(binValues) + 1
myHistogram = fltarr(nrBins)

nrBinValues = (size(binValues, /dimensions))[0]
if(nrBinValues eq 1) then stop

for tpoint = 0L, nrBinValues -1 do myHistogram[binValues[tpoint]]++

binValues = binsize*findgen( (size(data, /dimensions))[0])

if(self.invertBackground eq 1) then begin
backgroundColor = 16777215
foregroundColor  = 0
endif else begin
backgroundColor = 0
foregroundColor = 16777215
endelse

if(self.classMode eq 0) then pointColor = self.lineColor else begin
if(self.classType eq 1) then pointColor = self.class1Color else pointColor = self.class2Color
endelse

if(pointColor eq backgroundColor) then pointColor = foregroundColor
```

```
if(histShift gt -10) then minx = 0 else minx = histShift
if(overWrite eq 0) then begin
plot, [0], [0], xrange = [minx, maxx], yrange = [0, maxy], color = foregroundColor, background
= backgroundColor, xtitle = xlabel, ytitle = ylabel, charsize = 2, PSYM=3
endif
oplot, binValues+histShift*binsize, myHistogram/total(myHistogram), color = pointColor,
PSYM=10
END

PRO LVA_TraceData::PlotTumor, maxx, maxy, overWrite
time = (*self.timePtr)[self.timeindex]- (*self.timePtr)[0]
minutes = floor(time+0.1)
mysec = round((time - minutes)*60.)
timestring = string(minutes, format = '(I2)')+':'+ string(mysec, format = '(I02)')+ ' after
injection'

self->plotHistogram, (*self.tumorPtr)[*,self.timeindex], self.tumorBinSize, maxx, maxy,'SUV
'+timestring, 'Normalized count', overWrite
END

PRO LVA_TraceData::PlotTumorRatio, maxx, maxy, overWrite
time = (*self.timePtr)[self.timeindex]- (*self.timePtr)[0]
minutes = floor(time+0.1)
mysec = round((time - minutes)*60.)
timestring = string(minutes, format = '(I2)')+':'+ string(mysec, format = '(I02)')+ ' after
injection'
tissueMedian = self->LVA RefPercentile(0.5)
self->plotHistogram, (*self.tumorPtr)[*,self.timeindex]/tissueMedian[self.timeindex],
self.tumorBinSize, maxx, maxy, 'Tumor/tissue ratio '+timestring, 'Normalized count', overWrite
END

PRO LVA_TraceData::PlotPatlakSlope, maxx, maxy, overWrite
self->plotHistogram, *self.pSlPtr, self.patlakSlopeBinSize, maxx, maxy, 'Patlak slope',
'Normalized count', overWrite
END

PRO LVA_TraceData::PlotPatlakIntercept, maxx, maxy, overWrite
self->plotHistogram, *self.pInPtr, self.patlakInterceptBinSize, maxx, maxy, 'Patlak
intercept', 'Normalized count', overWrite
END

PRO LVA_TraceData::PlotMetabolicRate, maxx, maxy, overWrite
binValues = self.metabolicBinSize*findgen((size(*self.metabolicPtr, /dimensions))[0])
if(overWrite eq 0) then begin
plot, binvalues, *self.metabolicPtr/total(*self.metabolicPtr), xrange = [0, maxx], yrange =
[0, maxy], psym = 10
endif else begin
oplot, binvalues+0.0001*overWrite, *self.metabolicPtr/total(*self.metabolicPtr), color =
self.lineColor, psym = 10
endelse
END

PRO LVA_TraceData::PlotArrayPsymStyled, xarray, yarray, maxx, maxy, overWrite, _EXTRA = e
if(self.classMode eq 0) then begin
psymval = -((overWrite +3) MOD 6) -1
if(psymval eq -3) then psymval = -7
linStyVal = (overWrite) MOD 5
endif else begin
if(self.classType eq 1) then begin
psymval = -4
lineStyVal = 0
endif else begin
psymval = -5
lineStyVal = 0
endelse
endelse
if(overWrite eq 0) then begin
if(self.invertBackground eq 1) then begin
plot, xarray, yarray, xrange = [0, maxx], yrange = [0, maxy], color = 0, background =
16777215, charsize = 2, PSYM = -4, linestyle = linStyVal,  EXTRA = e
endif else plot, xarray, yarray, xrange = [0, maxx], yrange = [0, maxy], color = 16777215,
background = 0, charsize = 2, PSYM = -4, linestyle = linStyVal, _EXTRA = e
endif else begin
if(self.classMode eq 0) then drawColor = self.lineColor else begin
if(self.classType eq 1) then drawcolor = self.class1Color else drawcolor = self.class2Color
endelse
```

```
if(self.invertBackground eq 1) then begin
bc = 16777215
fc = 0
endif else begin
bc = 0
fc = 16777215
endelse
if(drawcolor eq bc) then drawcolor = fc
oplot, xarray, yarray, color = drawcolor, psym = psymval, linestyle = linStyVal, _EXTRA = e

endelse
END

PRO LVA_TraceData::PlotTumorMax, maxx, maxy, overWrite
pValues = fltarr((size(*self.tumorPtr, /dimensions))[1])
pValues = max(*self.tumorPtr, dimension = 1)*exp((*self.timePtr)*0.0063128)
self->PlotArrayPsymStyled, *self.timePtr, pValues, maxx, maxy, overWrite, xtitle =
'Time(min)', ytitle = 'Max SUV'
END

PRO LVA_TraceData::PlotTumorMaxScaled, maxx, maxy, overWrite
pValues = fltarr((size(*self.tumorPtr, /dimensions))[1])
pValues = max((*self.tumorPtr), dimension = 1)*exp((*self.timePtr)*0.0063128)/self-
>LVA_RefPercentile(0.5)
self->PlotArrayPsymStyled, *self.timePtr, pValues, maxx, maxy, overWrite, xtitle =
'Time(min)', ytitle = 'Max SUV/tissue ratio'
END

PRO LVA_TraceData::PlotTumorPeak, maxx, maxy, overWrite
pValues = fltarr((size(*self.tumorPeakPtr, /dimensions))[1])
pValues = max(*self.tumorPeakPtr, dimension = 1)*exp((*self.timePtr)*0.0063128)
self->PlotArrayPsymStyled, *self.timePtr, pValues, maxx, maxy, overWrite, xtitle =
'Time(min)', ytitle = 'Peak SUV'
END

PRO LVA_TraceData::PlotTumorPeakScaled, maxx, maxy, overWrite
pValues = fltarr((size(*self.tumorPeakPtr, /dimensions))[1])
pValues = max(*self.tumorPeakPtr, dimension = 1)*exp((*self.timePtr)*0.0063128)/self-
>LVA RefPercentile(0.5)
self->PlotArrayPsymStyled, *self.timePtr, pValues, maxx, maxy, overWrite, xtitle =
'Time(min)', ytitle = 'Peak SUV/tissue ratio'
END

PRO LVA_TraceData::PlotTumor90p, maxx, maxy, overWrite
pValues = fltarr((size(*self.tumorPtr, /dimensions))[1])
for tindex = 0, (size(*self.tumorPtr, /dimensions))[1]-1 do begin
sortedValues = (*self.tumorPtr)[sort((*self.tumorPtr)[*,tindex]),tindex]
indexPercentile = ((size(*self.tumorPtr, /dimensions))[0]-1.0)*0.98
pValues[tindex] = sortedValues[indexPercentile]*exp((*self.timePtr)[tindex]*0.0063128)
endfor
self->PlotArrayPsymStyled, *self.timePtr, pValues, maxx, maxy, overWrite, xtitle =
'Time(min)', ytitle = '98 percentile SUV'
END

PRO LVA_TraceData::PlotTumor90pScaled, maxx, maxy, overWrite
pValues = fltarr((size(*self.tumorPtr, /dimensions))[1])
for tindex = 0, (size(*self.tumorPtr, /dimensions))[1]-1 do begin
sortedValues = (*self.tumorPtr)[sort((*self.tumorPtr)[*,tindex]),tindex]
indexPercentile = ((size(*self.tumorPtr, /dimensions))[0]-1.0)*0.98
pValues[tindex] = sortedValues[indexPercentile]*exp((*self.timePtr)[tindex]*0.0063128)
endfor
pValues /= self->LVA RefPercentile(0.5)
self->PlotArrayPsymStyled, *self.timePtr, pValues, maxx, maxy, overWrite, xtitle =
'Time(min)', ytitle = '98 percentile SUV/tissue ratio'
END

PRO LVA_TraceData::PlotTumorMean, maxx, maxy, overWrite
medianValues = fltarr((size(*self.tumorPtr, /dimensions))[1])
for tindex = 0, (size(*self.tumorPtr, /dimensions))[1]-1 do begin
tumorMoments = MOMENT((*self.tumorPtr)[*, tindex])
medianValues[tindex] = tumorMoments[0]
endfor
self->PlotArrayPsymStyled, *self.timePtr, medianValues, maxx, maxy, overWrite, xtitle =
'Time(min)', ytitle = 'Mean SUV'
END
```

```
PRO LVA_TraceData::PlotTumorMeanScaled, maxx, maxy, overWrite
medianValues = fltarr((size(*self.tumorPtr, /dimensions))[1])
for tindex = 0, (size(*self.tumorPtr, /dimensions))[1]-1 do begin
tumorMoments = MOMENT((*self.tumorPtr)[*, tindex])
medianValues[tindex] = tumorMoments[0]
endfor
medianValues /= self->LVA_RefPercentile(0.5)
self->PlotArrayPsymStyled, *self.timePtr, medianValues, maxx, maxy, overWrite, xtitle =
'Time(min)', ytitle = 'Mean SUV/tissue ratio'
END

PRO LVA_TraceData::PlotTumorVariance, maxx, maxy, overWrite
variValues = fltarr((size(*self.tumorPtr, /dimensions))[1])
for tindex = 0, (size(*self.tumorPtr, /dimensions))[1]-1 do begin
tumorMoments = MOMENT((*self.tumorPtr)[*, tindex])
variValues[tindex] = tumorMoments[1]
endfor

if(overWrite eq 0) then begin
plot, *self.timePtr, variValues, xrange = [0, maxx], yrange = [0, maxy]
endif else oplot, *self.timePtr, variValues, color = self.lineColor
END

PRO LVA_TraceData::PlotTumorSkew, maxx, maxy, overWrite
skewValues = fltarr((size(*self.tumorPtr, /dimensions))[1])
for tindex = 0, (size(*self.tumorPtr, /dimensions))[1]-1 do begin
tumorMoments = MOMENT((*self.tumorPtr)[*, tindex])
skewValues[tindex] = tumorMoments[2]
endfor

if(overWrite eq 0) then begin
plot, *self.timePtr, skewValues, xrange = [0, maxx], yrange = [0, maxy]
endif else oplot, *self.timePtr, skewValues, color = self.lineColor
END

PRO LVA_TraceData::PlotTumorKurtosis, maxx, maxy, overWrite
kurtValues = fltarr((size(*self.tumorPtr, /dimensions))[1])
for tindex = 0, (size(*self.tumorPtr, /dimensions))[1]-1 do begin
tumorMoments = MOMENT((*self.tumorPtr)[*, tindex])
kurtValues[tindex] = tumorMoments[3]
endfor

if(overWrite eq 0) then begin
plot, *self.timePtr, kurtValues, xrange = [0, maxx], yrange = [0, maxy]
endif else oplot, *self.timePtr, kurtValues, color = self.lineColor
END

PRO LVA_TraceData::OPlotPlasma
;binValues = self.kBinSize*findgen( (size(*self.k1Ptr, /dimensions))[0])
oplot, *self.timePtr, *self.plasmaPtr, color = self.lineColor
END

PRO LVA_TraceData::OPlotMetabolicRate
binValues = self.metabolicBinSize*findgen((size(*self.metabolicPtr, /dimensions))[0])
oplot, binvalues, *self.metabolicPtr/total(*self.metabolicPtr), color = self.lineColor, psym =
10
END

PRO LVA_TraceData::Cleanup
self->LVA_EraseBuffers
END

PRO LVA_TraceData::LVA_EraseBuffers
if(ptr_valid(self.k1Ptr)      eq 1) then ptr_free, self.k1Ptr
if(ptr_valid(self.k2Ptr)      eq 1) then ptr_free, self.k2Ptr
if(ptr_valid(self.k3Ptr)      eq 1) then ptr_free, self.k3Ptr
if(ptr_valid(self.vbPtr)      eq 1) then ptr_free, self.vbPtr
if(ptr_valid(self.mbPtr)        eq 1) then ptr_free, self.mbPtr
if(ptr_valid(self.pearsonPtr)   eq 1) then ptr_free, self.pearsonPtr
if(ptr_valid(self.pSlPtr)       eq 1) then ptr_free, self.pSlPtr
if(ptr_valid(self.pInPtr)       eq 1) then ptr_free, self.pInPtr
if(ptr_valid(self.plasmaPtr)  eq 1) then ptr_free, self.plasmaPtr
if(ptr_valid(self.timePtr)           eq 1) then ptr_free, self.timePtr
if(ptr_valid(self.tumorPtr)   eq 1) then ptr_free, self.tumorPtr
if(ptr_valid(self.tissuePtr)  eq 1) then ptr_free, self.tissuePtr
if(ptr_valid(self.metabolicPtr) eq 1) then ptr_free, self.metabolicPtr
```

```
END
```

Scripts

```
PRO LVA_savePlasmaJpeg, slave, lvaBase, trcBase, pasName
    slave->LVA LoadState, lvaBase+pasName+'.lva'
    slave->LVA CalculatePlasmaAdaptation
    slave->LVA_PlotPlasma
    image = tvrd(true=1)
    write_jpeg, trcBase+ pasName +'(plasma).jpg', image, quality=100, true=1
    print, 'Built ' + pasName
END

PRO LVA_createTrcFile, slave, lvaBase, trcBase, pasName
    slave->LVA_LoadState, lvaBase+pasName+'.lva'
    slave->LVA_CalculateKSpace, 1
    slave->LVA CalculateR2Map
    slave->LVA CalculatePatlakMap, 0
    slave->LVA_SaveTrace, trcBase+pasName+'.trc'
    print, 'Built ' + pasName
END

PRO LVA_calculateAllTrace
    sliceDataPtr = OBJ NEW('LVA SliceData')
    sliceDataPtr->LVA_Constructor

    baseLvaDir = 'D:\SRC\'
    baseTrcDir = 'D:\tracefiles\'

    pasIds = strarr(22)
    pasIds[0] = 'pas0sek1'
    pasIds[1] = 'pas0sek2'
    ...
    pasIds[21] = 'pas10sek1'

    for index = 0, (size(pasIds, /dimensions))[0]-1 do begin
        LVA_createTrcFile, sliceDataPtr, baseLvaDir, baseTrcDir, pasIds[index]
    endfor

    OBJ_DESTROY, sliceDataPtr
END

PRO LVA_saveWHKJpeg, slave, renderWindow,  lvaBase, trcBase, pasName, frame
    slave->LVA LoadState, lvaBase+pasName+'.lva'
    slave->LVA_SetDepthIndex, frame

    tWin = slave->LVA_GetTumorOutputWindow()
    oWin = slave->LVA_GetOutputWindow()

    outputSUVdata = 0
    outputPharmadata = 0
    outputPatlak = 1

    if(outputSUVdata eq 1) then begin
        slave->LVA_DisplayCTSlice
        slave->LVA BlendTumorOntoFrame, 9, renderWindow
        image = tvrd(true=1)
        write jpeg, trcBase+ pasName + 'tumorCT' + '.jpg', image[*,tWin[0]:(tWin[0]+tWin[2]),
       tWin[1]:(tWin[1]+tWin[3])], quality=100, true=1

        slave->LVA_SetTimeIndex, 1
        slave->LVA DisplaySUVBlended
        slave->LVA_BlendEdgeTumorOntoFrame, 9, renderWindow
        image = tvrd(true=1)
        write_jpeg, trcBase+ pasName + 'Pet0.2min' + '.jpg',
       image[*,oWin[0]:(oWin[0]+oWin[2]), oWin[1]:(oWin[1]+oWin[3])], quality=100, true=1

        slave->LVA SetTimeIndex, 2
        slave->LVA_DisplaySUVBlended
        slave->LVA_BlendEdgeTumorOntoFrame, 9, renderWindow
        image = tvrd(true=1)
        write_jpeg, trcBase+ pasName + 'Pet0.5min' + '.jpg',
```

```
    image[*,oWin[0]:(oWin[0]+oWin[2]), oWin[1]:(oWin[1]+oWin[3])], quality=100, true=1

     slave->LVA SetTimeIndex, 3
     slave->LVA DisplaySUVBlended
     slave->LVA BlendEdgeTumorOntoFrame, 9, renderWindow
     image = tvrd(true=1)
     write_jpeg, trcBase+ pasName + 'Pet0.75min' + '.jpg',
    image[*,oWin[0]:(oWin[0]+oWin[2]), oWin[1]:(oWin[1]+oWin[3])], quality=100, true=1

     slave->LVA SetTimeIndex, 4
     slave->LVA ToggleSUVTimeAveraging
     slave->LVA_DisplaySUVBlended
     slave->LVA_BlendEdgeTumorOntoFrame, 9, renderWindow
     image = tvrd(true=1)
     write jpeg, trcBase+ pasName + 'Pet1min' + '.jpg', image[*,oWin[0]:(oWin[0]+oWin[2]),
    oWin[1]:(oWin[1]+oWin[3])], quality=100, true=1

     slave->LVA_SetTimeIndex, 8
     slave->LVA_ToggleSUVTimeAveraging
     slave->LVA DisplaySUVBlended
     slave->LVA BlendEdgeTumorOntoFrame, 9, renderWindow
     image = tvrd(true=1)
     write_jpeg, trcBase+ pasName + 'Pet2min' + '.jpg', image[*,oWin[0]:(oWin[0]+oWin[2]),
    oWin[1]:(oWin[1]+oWin[3])], quality=100, true=1

     slave->LVA SetTimeIndex, 14
     slave->LVA DisplaySUVBlended
     slave->LVA BlendEdgeTumorOntoFrame, 9, renderWindow
     image = tvrd(true=1)
     write_jpeg, trcBase+ pasName + 'Pet4min' + '.jpg', image[*,oWin[0]:(oWin[0]+oWin[2]),
    oWin[1]:(oWin[1]+oWin[3])], quality=100, true=1

     slave->LVA SetTimeIndex, 22
     slave->LVA DisplaySUVBlended
     slave->LVA_BlendEdgeTumorOntoFrame, 9, renderWindow
     image = tvrd(true=1)
     write_jpeg, trcBase+ pasName + 'Pet8min' + '.jpg', image[*,oWin[0]:(oWin[0]+oWin[2]),
    oWin[1]:(oWin[1]+oWin[3])], quality=100, true=1

     slave->LVA SetTimeIndex, 31
     slave->LVA_DisplaySUVBlended
     slave->LVA_BlendEdgeTumorOntoFrame, 9, renderWindow
     image = tvrd(true=1)
     write jpeg, trcBase+ pasName + 'Pet17min' + '.jpg', image[*,oWin[0]:(oWin[0]+oWin[2]),
    oWin[1]:(oWin[1]+oWin[3])], quality=100, true=1

     slave->LVA_SetTimeIndex, 44
     slave->LVA_ToggleSUVTimeAveraging
     slave->LVA DisplaySUVBlended
     slave->LVA BlendEdgeTumorOntoFrame, 9, renderWindow
     image = tvrd(true=1)
     write_jpeg, trcBase+ pasName + 'Pet43min' + '.jpg', image[*,oWin[0]:(oWin[0]+oWin[2]),
    oWin[1]:(oWin[1]+oWin[3])], quality=100, true=1
endif

if(outputPharmadata eq 1) then begin
     slave->LVA_CalculateKSpace, 1, 1
     slave->LVA_DisplayKComposite, 1
     slave->LVA BlendEdgeTumorOntoFrame, 9, renderWindow
     image = tvrd(true=1)
     write jpeg, trcBase+ pasName + 'k1' + '.jpg', image[*,oWin[0]:(oWin[0]+oWin[2]),
    oWin[1]:(oWin[1]+oWin[3])], quality=100, true=1
     slave->LVA_DisplayKComposite, 2
     slave->LVA_BlendEdgeTumorOntoFrame, 9, renderWindow
     image = tvrd(true=1)
     write jpeg, trcBase+ pasName + 'k2' + '.jpg', image[*,oWin[0]:(oWin[0]+oWin[2]),
    oWin[1]:(oWin[1]+oWin[3])], quality=100, true=1
     slave->LVA_DisplayKComposite, 3
     slave->LVA_BlendEdgeTumorOntoFrame, 9, renderWindow
     image = tvrd(true=1)
     write jpeg, trcBase+ pasName + 'k3' + '.jpg', image[*,oWin[0]:(oWin[0]+oWin[2]),
    oWin[1]:(oWin[1]+oWin[3])], quality=100, true=1
     slave->LVA_DisplayMetabolicComposite
     slave->LVA_BlendEdgeTumorOntoFrame, 9, renderWindow
     image = tvrd(true=1)
     write_jpeg, trcBase+ pasName + 'metabolic' + '.jpg',
```

```
        image[*,oWin[0]:(oWin[0]+oWin[2]), oWin[1]:(oWin[1]+oWin[3])], quality=100, true=1
         slave->LVA_DisplayVBComposite
         slave->LVA BlendEdgeTumorOntoFrame, 9, renderWindow
         image = tvrd(true=1)
         write jpeg, trcBase+ pasName + 'vb' + '.jpg', image[*,oWin[0]:(oWin[0]+oWin[2]),
oWin[1]:(oWin[1]+oWin[3])], quality=100, true=1
    endif

    if(outputPatlak eq 1) then begin
        case pasName of
            'pas0sek1' : begin
                slave->LVA_SetMinDisplayValue, 30 , 6
                slave->LVA_SetMaxDisplayValue, 370 , 6
                slave->LVA_SetMinDisplayValue, 100 , 7
                slave->LVA_SetMaxDisplayValue, 885 , 7
             endcase
             …
             'pas10sek1' : begin
                slave->LVA_SetMinDisplayValue, 20 , 6
                slave->LVA SetMaxDisplayValue, 80 , 6
                slave->LVA SetMinDisplayValue, 135 , 7
                slave->LVA_SetMaxDisplayValue, 543 , 7
             endcase
             else :
        endcase

        slave->LVA CalculatePatlakMap, 1
        slave->LVA DisplayPatlakSlopeSlice
        slave->LVA_BlendEdgeTumorOntoFrame, 9, renderWindow
        image = tvrd(true=1)
        write_jpeg, trcBase+ pasName + 'newpatslope' + '.jpg',
        image[*,oWin[0]:(oWin[0]+oWin[2]), oWin[1]:(oWin[1]+oWin[3])], quality=100, true=1

        slave->LVA DisplayPatlakInterceptSlice
        slave->LVA_BlendEdgeTumorOntoFrame, 9, renderWindow
        image = tvrd(true=1)
        write_jpeg, trcBase+ pasName + 'newpatinter' + '.jpg',
        image[*,oWin[0]:(oWin[0]+oWin[2]), oWin[1]:(oWin[1]+oWin[3])], quality=100, true=1
    endif

    print, 'Built ' + pasName
END

PRO LVA_calculateHB
    sliceDataPtr = OBJ_NEW('LVA_SliceData')
    sliceDataPtr->LVA_Constructor

    baseLvaDir = 'D:\SRC\'
    baseTrcDir = 'D:\outputfiles\Images\Pasient 1\'
    baseTrcDir = 'D:\outputfiles\Images\'

    pasDepth = fltarr(22)
    pasIds = strarr(22, 2)
    pasIds[0,0] = 'pas0sek1'
    pasIds[0,1] = 'Pasient01'
    pasDepth[0] = 22
    pasIds[1,0] = 'pas0sek2'
    pasIds[1,1] = 'Pasient02'
    pasDepth[1] = 37
    ...
    pasIds[21,0] = 'pas10sek1'
    pasIds[21,1] = 'Pasient11'
    pasDepth[21] = 7


    Window, XSIZE = 512, YSIZE = 512, /FREE, /PIXMAP
    renderWindow = !D.Window
    WSet, renderWindow

    sliceDataPtr->LVA TogglePetSmoothing
    notherapy = [indgen(5), indgen(4)+11, 17, 21]
    for index = 0, 21 do begin
        if(index eq 1) then LVA_saveWHKJpeg, sliceDataPtr, renderWindow, baseLvaDir,
       baseTrcDir+ pasIds[index,1] +'\', pasIds[index,0], pasDepth[index]
        ;if(where(notherapy eq index) ne -1) then LVA_saveWHKJpeg, sliceDataPtr, renderWindow,
         baseLvaDir, baseTrcDir+ pasIds[index,1] +'\', pasIds[index,0], pasDepth[index]
```

```
        ;LVA_saveWHKJpeg, sliceDataPtr, renderWindow, baseLvaDir, baseTrcDir+ pasIds[index,1]
+'\', pasIds[index,0], pasDepth[index]
    endfor

    OBJ_DESTROY, sliceDataPtr
END

PRO LVA_printResiduals, pasIds, sliceDataPtr, inputBaseDir, outputBaseDir
     tumResiduals = ptrArr(n_elements(pasIds))
    plasResiduals = ptrArr(n_elements(pasIds))
    timearrs = ptrArr(n_elements(pasIds))

    for index = 0, n_elements(pasIds)-1 do begin
        sliceDataPtr->LVA_LoadState, inputBaseDir + pasIds[index] + '.lva'
        sliceDataPtr->LVA_CalculateKSpace, 1, 0
        plasr = sliceDataPtr->LVA_GetPlasmaSUVResiduals()
        tumr = sliceDataPtr->LVA_GetTumorSUVResiduals()
        tarr = sliceDataPtr->LVA_GetOriginalTimeArray()
        tumResiduals[index] = ptr_new(fltarr(n_elements(tumr)))
        *tumResiduals[index] = tumr
        plasResiduals[index] = ptr_new(fltarr(n_elements(plasr)))
        *plasResiduals[index] = plasr
        timearrs[index] = ptr_new(fltarr(n_elements(tarr)))
        *timearrs[index] = tarr

        plot, tarr, plasr, xtitle = 'Time (min)', ytitle = pasIds[index] +
       ' plasma SUV residual', charsize = 2, background = 2L^24L -1, color = 0
        image = tvrd(true=1)
        write_jpeg, outputBaseDir+ pasIds[index] + 'plasresidual' + '.jpg', image,
       quality=100, true=1

        plot, tarr, tumr, xtitle = 'Time (min)', ytitle = pasIds[index] +
       'tumor SUV residual', charsize = 2, background = 2L^24L -1, color = 0
        image = tvrd(true=1)
        write_jpeg, outputBaseDir+ pasIds[index] + 'tumorresidual' + '.jpg', image,
       quality=100, true=1
    endfor
    sumarr = fltarr(n elements(*plasResiduals[0]))
    for ind = 0, n_elements(pasIds)-1 do begin
        residualInterpol  = interpol(*plasResiduals[ind]/float(n_elements(pasIds)),
         *timearrs[ind], *timearrs[0])
        sumarr += residualInterpol
    endfor
    plot, *timearrs[0], sumarr, xtitle = 'Time (min)', ytitle = 'Plasma SUV residual',
charsize = 2, background = 2L^24L -1, color = 0
    image = tvrd(true=1)
    write_jpeg, outputBaseDir+ 'totalplasmaresidual' + '.jpg', image, quality=100, true=1

    sumarr = fltarr(n elements(*tumResiduals[0]))
    for ind = 0, n_elements(pasIds)-1 do begin
        residualInterpol  = interpol(*tumResiduals[ind]/float(n_elements(pasIds)),
*timearrs[ind], *timearrs[0])
        sumarr += residualInterpol
    endfor
    plot, *timearrs[0], sumarr, xtitle = 'Time (min)', ytitle = 'Tumor SUV residual',
       charsize = 2, background = 2L^24L -1, color = 0
    image = tvrd(true=1)
    write_jpeg, outputBaseDir + 'totaltumorresidual' + '.jpg', image, quality=100, true=1
    ;stop
    for ind = 0, n_elements(tumResiduals) -1 do ptr_free, tumResiduals[ind]
    for ind = 0, n_elements(plasResiduals) -1 do ptr_free, plasResiduals[ind]
    for ind = 0, n_elements(timearrs) -1 do ptr_free, timearrs[ind]
END

PRO LVA_printNoiseResponse, pasIds, sliceDataPtr, inputBaseDir, outputBaseDir
    for index = 0, n_elements(pasIds)-1 do begin
        sliceDataPtr->LVA_LoadState, inputBaseDir + pasIds[index] + '.lva'
        print, pasIds[index], ': ', sliceDataPtr->CheckNoiseResponse()
    endfor
END

PRO LVA_printParamterResponse, pasIds, sliceDataPtr, inputBaseDir, outputBaseDir
    for index = 5, n_elements(pasIds)-1 do begin
        sliceDataPtr->LVA_LoadState, inputBaseDir + pasIds[index] + '.lva'
        print, pasIds[index], ': ', sliceDataPtr->LVA_CheckKSpace(1)
   endfor
```

```
END

PRO LVA_calculateImpulseResponse
    sliceDataPtr = OBJ_NEW('LVA SliceData')
    sliceDataPtr->LVA_Constructor

    baseLvaDir = 'D:\SRC\'
    baseTrcDir = 'D:\outputfiles\residuals\'

    pasIds = strarr(11)
    pasIds[0] = 'pas0sek1'
    pasIds[1] = 'pas0sek2'
    …
    pasIds[10] = 'pas10sek1'

    Window, XSIZE = 512, YSIZE = 512, /FREE
    renderWindow = !D.Window
    WSet, renderWindow

    ;LVA printResiduals, pasIds, sliceDataPtr, baseLvaDir, baseTrcDir
    LVA printNoiseResponse, pasIds, sliceDataPtr,  baseLvaDir, baseTrcDir
    ;LVA_printParamterResponse, pasIds, sliceDataPtr,  baseLvaDir, baseTrcDir

    WDelete, renderWindow
END

PRO LVA_savePatlakJpeg, slave, lvaBase, trcBase, pasName
    slave->LVA LoadState, lvaBase+pasName+'.lva'
    slave->LVA_PlotTumorPlasmaRatio
    image = tvrd(true=1)
    write_jpeg, trcBase+ pasName +'(patlak).jpg', image, quality=100, true=1
    print, 'Built ' + pasName
END

PRO LVA_calculatePatlak
    sliceDataPtr = OBJ_NEW('LVA_SliceData')
    sliceDataPtr->LVA_Constructor

    baseLvaDir = 'D:\SRC\'
    baseTrcDir = 'D:\outputfiles\patlakfiles\'

    pasIds = strarr(22)
    pasIds[0] = 'pas0sek1'
    pasIds[1] = 'pas0sek2'
    ...
    pasIds[21] = 'pas10sek1'


    Window, XSIZE = 512, YSIZE = 512, /FREE, /PIXMAP
    renderWindow = !D.Window
    WSet, renderWindow

    for index = 0, (size(pasIds, /dimensions))[0]-1 do begin
        LVA_savePatlakJpeg, sliceDataPtr, baseLvaDir, baseTrcDir, pasIds[index]
    endfor
    WDelete, renderWindow
    OBJ_DESTROY, sliceDataPtr
END

PRO LVA_AnalyzerOutput
    slave = OBJ NEW('LVA TraceAnalyzer')
    slave->LVA_Constructor

    baseInputDir = 'D:\tracefilesSmoothed\'
    baseOutputDir = 'D:\outputfiles\analyzerfiles\'

    pasIds = strarr(11)
    pasIds[0] = 'pas0sek1'
    pasIds[1] = 'pas0sek2'
    …
    pasIds[10] = 'pas10sek1'

    Window, XSIZE = 600, YSIZE = 600, /FREE, /PIXMAP
    renderWindow = !D.Window
    WSet, renderWindow
```

```
    for index = 0, (size(pasIds, /dimensions))[0]-1 do begin
        slave->AddTrace, baseInputDir+pasIds[index]+'.trc', 0
    endfor

    slave->InvertBackground
    slave->SetXScatterMode, 1, [7,0,0]
    slave->SetYScatterMode, 1, [8,0,0]
    slave->DrawFrame, 31
    write_jpeg, baseOutputDir +'patlakmedian.jpg', tvrd(true=1), quality=100, true=1

    slave->SetXScatterMode, 1, [7,2,0]
    slave->SetYScatterMode, 1, [8,2,0]
    slave->DrawFrame, 31
    write_jpeg, baseOutputDir +'patlakvariance.jpg', tvrd(true=1), quality=100, true=1

    slave->PlotUmatrix, 60, 500
    write_jpeg, baseOutputDir +'umatrix.jpg', tvrd(true=1), quality=100, true=1
    OBJ_DESTROY, slave

    slave = OBJ_NEW('LVA TraceAnalyzer')
    baseInputDir = 'D:\Archive\tracefiles110714\'
    for index = 0, (size(pasIds, /dimensions))[0]-1 do begin
        slave->AddTrace, baseInputDir+pasIds[index]+'.trc', 0
    endfor
    slave->PlotUmatrix, 60, 500
    write jpeg, baseOutputDir +'umatrixold.jpg', tvrd(true=1), quality=100, true=1
    OBJ DESTROY, slave
    WDelete, renderWindow
    print, 'Analyzer done'
END

PRO LVA_AnalyzerScatters
    slave = OBJ NEW('LVA TraceAnalyzer')
    slave->LVA_Constructor

    baseInputDir = 'D:\tracefiles\'
    baseOutputDir = 'D:\outputfiles\patient scatter\'

    pasIds = strarr(11)
    pasIds[0] = 'pas0sek1'
    pasIds[1] = 'pas0sek2'
    …
    pasIds[10] = 'pas10sek1'

    Window, XSIZE = 600, YSIZE = 600, /FREE, /PIXMAP
    renderWindow = !D.Window
    WSet, renderWindow

    for index = 0, (size(pasIds, /dimensions))[0]-1 do begin
        slave->AddTrace, baseInputDir+pasIds[index]+'.trc', 0
    endfor

    slave->InvertBackground
    for pasind = 0, n_elements(pasIds)-1 do begin
        slave->SetXScatterMode, 1, [pasind,0,0]
        slave->SetYScatterMode, 1, [pasind,8,0]
        slave->DrawFrame, 31
        write_jpeg, baseOutputDir + 'pas' +string(pasind+1, format = '(I2)')+'suv2vb'+'.jpg',
       tvrd(true=1), quality=100, true=1
       …
        slave->SetXScatterMode, 1, [pasind,4,0]
        slave->SetYScatterMode, 1, [pasind,5,0]
        slave->DrawFrame, 31
        write_jpeg, baseOutputDir + 'pas' +string(pasind+1, format = '(I2)')+'k3met'+'.jpg',
       tvrd(true=1), quality=100, true=1
    endfor

    OBJ_DESTROY, slave
    WDelete, renderWindow
    print, 'Scatter done'
END
```